UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALGORITMO DE ENXAME DE PARTÍCULAS PARA RESOLUÇÃO DO PROBLEMA DA PROGRAMAÇÃO DA PRODUÇÃO *JOB-SHOP* FLEXÍVEL MULTIOBJETIVO

GABRIEL DIEGO DE AGUIAR ARANHA

ORIENTADOR: PROF. DR. EDILSON REIS RODRIGUES KATO

São Carlos - SP
Julho/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALGORITMO DE ENXAME DE PARTÍCULAS PARA RESOLUÇÃO DO PROBLEMA DA PROGRAMAÇÃO DA PRODUÇÃO *JOB-SHOP* FLEXÍVEL MULTIOBJETIVO

GABRIEL DIEGO DE AGUIAR ARANHA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Inteligência Artificial
Orientador: Dr. Edilson Reis Rodrigues Kato

São Carlos - SP
Julho/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Gabriel Diego de Aguiar Aranha, realizada em 19/08/2016.

Prof. Dr. Edilson Reis Rodrigues Kato
(UFSCar)

Prof. Dr. Roberto Hideaki Tsunaki
(EESC-USP)

.....

Prof. Dr. Paulo Eigi Miyagi
(EPUSP)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Prof. Dr. Paulo Eigi Miyagi . Depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Gabriel Diego de Aguiar Aranha.

Prof. Dr. Edilson Reis Rodrigues Kato
Coordenadora da Comissão Examinadora
(UFSCar)

RESUMO

As empresas atualmente buscam meios de ampliarem suas vantagens competitivas, otimizando sua produção, e neste contexto, encontraram soluções nas atividades de programação da produção. A programação da produção do tipo *job-shop*, resulta em um dos problemas mais complexos de combinação, o *Job-shop Scheduling Problem* (JSP), cuja resolução determinística é inviável em tempo computacional polinomial. O *Flexible Job-shop Scheduling Problem* (FJSP) é uma extensão do clássico JSP e tem sido amplamente relatado na literatura. Desta forma, algoritmos de otimização têm sido desenvolvidos e avaliados nas últimas décadas, com o intuito de fornecer planejamentos de produção mais eficientes, com destaque para os algoritmos de inteligência artificial do tipo enxame, que nas pesquisas mais recentes obtiveram resultados satisfatórios. O FJSP permite que uma operação seja processada por qualquer recurso produtivo advindo de um conjunto de recursos ao longo de diferentes roteiros. Este problema é comumente desmembrado em dois subproblemas, a atribuição de recursos para as operações, que é chamado de roteamento, e programação das operações. No contexto do FJSP, a proposta dessa pesquisa apresenta a resolução do FJSP em caráter multiobjetivo, utilizando a abordagem hierárquica, que divide o problema em dois subproblemas, sendo o Enxame de Partículas (PSO), responsável pela resolução do subproblema de roteamento e incumbindo três algoritmos de busca local, Reinício Aleatório de Subida de Colina (RRHC), Arrefecimento Simulado (SA) e Busca Tabu (TS), para a resolução do subproblema de programação. A implementação do algoritmo proposto, dispõe de novas estratégias na inicialização da população, deslocamento das partículas, alocação estocástica das operações e tratamento de cenários parcialmente flexíveis. Resultados experimentais obtidos em base de testes comumente usada, comprovam a eficácia da hibridização proposta, e a vantagem do algoritmo RRHC em relação aos outros na resolução do subproblema de programação.

Palavras-chave: Enxame de Partículas, Flexible Job Shop Scheduling Problem, Arrefecimento Simulado, Busca Tabu, Reinício Aleatório de Subida de Colina, otimização multiobjetivo, roteamento, programação.

ABSTRACT

The companies today are looking for ways to expand their competitive advantages, optimizing their production, and in this context, they found solutions in activities of production scheduling. The production scheduling of the type job-shop, results in one of the most complex problems of combination, the Job-shop Scheduling Problem (JSP), which deterministic resolution is not feasible in polynomial computational time. The Flexible Job-shop Scheduling Problem (FJSP) is a classic extension of the JSP and has been widely reported in the literature. Thus, optimization algorithms have been developed and evaluated in the last decades, in order to provide more efficient production planning, with emphasis to artificial intelligence algorithms of the swarm type, that the latest research presented favorable results. The FJSP allows an operation to be processed for any machine arising from a set of machines along different routes. This problem is commonly dismembered into two sub-problems, the assignment of machines for operations, which is called routing, and operation scheduling. In the FJSP context, this research presents the resolution of the FJSP multi-objective, using a hierarchical approach that divides the problem into two subproblems, being the Particle Swarm Optimization (PSO), responsible for resolving the routing sub-problem, and tasking three local search algorithms, Random Restart Hill Climbing (RRHC), Simulated Annealing (SA) and Tabu Search (TS), for the resolution of scheduling sub-problem. The implementation of the proposed algorithm has new strategies in the population initialization, displacement of particles, stochastic allocation of operations, and management of scenarios partially flexible. Experimental results using technical benchmarks problems are conducted, and proved the effectiveness of the hybridization, and the advantage of RRHC algorithm compared to others in the resolution of the scheduling subproblem.

Keywords: Particle Swarm, Flexible Job Shop Scheduling Problem, Simulated Annealing, Random Restart Hill Climbing, Tabu Search, optimization multi-objective, routing, scheduling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FJSP – *Flexible Job-shop Scheduling Problem*

GA – *Genetic Algorithm*

JSP – *Job-shop Scheduling Problem*

LS – *Local Search*

PCP – *Planejamento e Controle da Produção*

P-FJSP – *Partial Flexible Job-shop Scheduling Problem*

PSO – *Particle Swarm Optimization*

RRHC – *Random Restart Hill Climbing*

SA – *Simulates Annealing*

SPT – *Shortest Processing Time*

T-FJSP – *Total Flexible Job-shop Scheduling Problem*

TS – *Tabu Search*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contexto

A aspiração por aumento da produtividade no mundo empresarial moderno tem estimulado as empresas a procurar oportunidades cada vez mais precisas e inexploradas para a redução de custos e geração de lucros (ROSHANAEI; ELMARAGHY, 2012). Destacam-se também os conceitos contemporâneos sobre responsabilidade socioambiental, uma vez que os consumidores estão mais conscientes, fazendo com que as empresas optem por soluções mais concisas, planejadas e com menos consumo. Uma opção viável para características de mercado como essas pode ser a adoção de técnicas mais sofisticadas para o Planejamento e Controle da Produção (PCP), capazes de aproveitar da melhor maneira possível a capacidade produtiva dos recursos disponíveis (HEINONEN; PETTERSSON, 2007).

No decorrer das últimas seis décadas, mecanismos eficientes de PCP foram reconhecidos como alternativa para o aumento da produtividade e utilização dos recursos (ROSHANAEI; ELMARAGHY, 2012). Nesse âmbito, a programação de produção de múltiplos recursos produtivos em lote (*job-shop*) vem desencadeando cada vez mais estudos, quando observados as progressivas necessidades de flexibilização e automação exigidas pela dinâmica concorrência produtiva contemporânea, as quais tornam as atividades de programação específica da produção progressivamente complexas (KATHAWALA; ALLEN, 1993), caracterizando assim no *Job-shop Scheduling Problem* (JSP). O JSP tem como premissa minimizar os custos de produção no todo (tempo, dinheiro, recursos humanos), onde conjuntos

de operações são designados a serem processados em um conjunto de recursos produtivos (máquinas, linhas de produção, pessoal/mão-de-obra, ferramentas, etc.), cada operação necessita de uma sequência específica de recursos produtivos conhecidos antecipadamente, cada recurso processa iterativamente operações que chegam até ela, uma por vez sem interrupção, resultando assim na programação da produção. Almeja-se comumente como objetivo a atribuição do maior número plausível de tarefas (operações) aos recursos disponíveis, no menor tempo de processamento possível.

Embora o conceito JSP possa ser explanado facilmente, é muito custoso computacionalmente, ele está enquadrado nos problemas NP-difícil, isto é, aqueles que não podem ser deterministicamente resolvidos em tempo computacional polinomial (GAREY; JOHNSON; SETHI, 1976). Em especial pelo motivo desta técnica apresentar um dos mais difíceis problemas de otimização combinatória abordados pela teoria clássica de programação (GAREY; JOHNSON, 1979).

Flexible Job-shop Scheduling Problem (FJSP) é uma extensão do JSP referido que permite uma operação ser processada por um recurso advindo de um conjunto de recursos alternativos, propiciando uma maior proximidade da situação real das fábricas. Devido às necessidades adicionais para determinar a atribuição e combinação de operações nos recursos, o FJSP é mais complexo que o JSP, tendo também todas as dificuldades do JSP (ZHANG et al., 2009)

O FJSP pode ser dividido em dois subproblemas principais, o de roteamento e o de programação. O subproblema de roteamento atribui a cada operação um recurso dentre um conjunto de recursos autorizados para a respectiva operação. O subproblema de programação envolve a programação das operações atribuídas aos recursos, a fim de se obter um cronograma viável que minimiza um objetivo pré-definido (XIA; WU, 2005). Para resolução de tais problemas, são possíveis, segundo DORIGO; BLUM (2005), dois métodos de otimização global, quais sejam: completas e aproximadas. Uma vez que as técnicas completas resultam em soluções ótimas em tempo ágil, as aproximadas anseiam soluções que atendam às necessidades sem uma otimalidade, na maioria dos casos para economizar custo computacional. Nessa resolução se enquadram algoritmos híbridos que somam a performance global mais comumente visto na geração da população (roteiros), e local no âmbito de minuciar a melhor programação dos roteiros.

Por conseguinte, meta-heurísticas que possuem a finalidade de otimização são indicadas para problemas dessa ordem. O algoritmo Enxame de Partículas (do inglês *Particle Swarm Optimization* (PSO)), proporciona uma otimização global simulando o comportamento social humano e de algumas espécies de animais, onde a troca de informações o faz percorrer o espaço de soluções sendo influenciado pela experiência de um grupo de indivíduos pequeno e de toda a população. Esta iterativa comparação de experiências, e o fato de que cada indivíduo possui autonomia para a exploração das soluções, evidenciam a aplicabilidade bem-sucedida como busca global, e a motivação do uso desta técnica na resolução do subproblema de roteamento. O PSO é um algoritmo adaptável, permitindo assim que sua função de aptidão seja gerada por outras meta-heurísticas, no caso desta pesquisa, a avaliação dos objetivos que são feitos por uma busca local.

A resolução do subproblema de programação no escopo deste trabalho tem como objetivo principal a aquisição do menor tempo de execução de todas as operação, resultado da permutação de combinações do roteiro gerado pelo PSO. Assim, é necessário um algoritmo preciso e rápido, como é o caso dos algoritmos *Simulated Annealing* (SA), um algoritmo análogo conceitualmente ao processo de arrefecimento e cristalização de corpos sólidos e o *Tabu Search* (TS), um algoritmo que evita movimentos já tomados (tabus) em busca de uma otimização, ambos algoritmos são amplamente encontrados com este escopo na literatura. Outro algoritmo preciso e rápido é o *Random Restart Hill Climbing* (RRHC), porém este não é muito utilizado neste contexto, trate-se de um modelo matemático que apenas aceita uma nova solução caso ela seja melhor que a anterior, o que o torna um algoritmo guloso, com o adicional de que passadas uma quantidade de iterações, ele reinicia sua posição inicial e recomeça busca.

1.2 Motivação

Planejar e controlar a produção tem gerado bastante divagação sobre como ser devidamente aplicado em ferramentas para sistemas de fabricação, devido à crescente demanda dos consumidores por variedade, ciclos de vida dos produtos reduzidos, mudança de mercado com concorrência global, e rápido desenvolvimento

de novos processos e tecnologias (TAY; HO, 2008). Esse conceito contemporâneo da economia global vem impondo novos desafios à indústria baseados na evolução do processo de manufatura de um produto, o qual possui o âmago na qualidade, adaptação e tempo para ser produzido (EL MARAGHY, 2006).

Diante da realidade supracitada, novas técnicas e métodos de PCP são propostos e desenvolvidos cientificamente para auxiliar a evolução do setor industrial, mais especificadamente de empresas que tem sua produção arranjada na forma de um *job-shop.* De acordo com (XIA; WU, 2005), essas empresas lidam com problemas consideravelmente difíceis, mesmo com esforços computacionais, pois a solução gerada a partir das combinações de roteiros e programação não é alcançada deterministicamente em tempo polinomial (NP-difícil), isto é, modelos matemáticos clássicos que buscam soluções ótimas não são indicados para lidar com tais cenários. Porém as complexidades são ainda maiores se adicionados múltiplos critérios de desempenho, recursos paralelos que possam processar as mesmas operações e restrições de precedência.

Conforme ZITZLER (1999), COELLO (2001), COLLETTE; SIARRY (2002) e ZITZLER; LAUMANNS; BLEULER (2004), algoritmos de inteligência artificial e bioinspirado vêm adquirindo reconhecimento por conta de eficientes resultados na otimização de JSP. Tendo como precursor BRANDIMARTE (1993), que elaborou o primeiro modelo de resolução de um FJSP utilizando a meta-heurística *Tabu Search* (TS). Desde então são propostas meta-heurísticas que possam em tempo ágil solucionar o FJSP com, em sua maioria, restrições e condições comuns às empresas de *job-shop,* porém como os tempos de convergência variam de acordo com a meta-heurística selecionada e as complexidades impostas, a pesquisa sobre este campo é assegurada, tanto para melhorar os resultados atuais quanto para propor soluções com abordagens mais complexas.

Dado que os recursos produtivos e a complexidade de fabricação de uma empresa só crescem, faz-se necessário a utilização de um algoritmo capaz de abranger todo o espaço de solução possível, pois segundo MOSLEHI; MAHNAM (2011), o PSO se destaca por ser capaz de percorrer este espaço, mesmo que extenso, utilizando uma busca local para a programação correta dos roteamentos de operações encontrados, com o objetivo de diminuir o *makespan* - tempo de completude do início da primeira operação até a última operação, para um cenário

proposto, podendo contribuir com melhores resultados em comparação com outras técnicas (BAGHERI et al., 2010).

1.3 Objetivos

Esse trabalho tem como objetivo principal propor a resolução do FJSP com a utilização de PSO de forma híbrida com um algoritmo de busca local. Também será realizado um comparativo entre três meta-heurísticas auxiliares. Duas delas, SA e TS, são amplamente recomendadas para o subproblema de programação, segundo a literatura. Já o RRHC é proposto como uma terceira alternativa. Para validar a hibridização do PSO com as meta-heurísticas esta pesquisa utiliza três critérios a serem minimizados do FJSP, caracterizando assim a formulação multiobjetivo, com base na programação da produção, considerando restrições abordadas na literatura e comparando os resultados com os já obtidos no segmento definidos na proposta. Os critérios são:

(1) Makespan C_m: O tempo de conclusão máxima dos recursos, isto é, o *makespan*, avaliando o impacto na variação dos parâmetros, tal como a implantação de mecanismos que visam agregar melhoria para o algoritmo PSO;

(2) Workload machine W_m: A carga de trabalho máxima de um recurso isolado, isto é, o máximo de tempo de produção gasto em qualquer recurso, garantindo uma resolução balanceada; e

(3) Workload total W_t: O volume total de trabalho dos recursos, o que representa o total tempo de trabalho sobre todos os recursos, uma vez que mesmo com um *makespan* baixo os recursos podem ter trabalhado quase incessantemente em paralelo.

1.4 Organização do Trabalho

Além do capítulo introdutório esta pesquisa está subdivida conforme os capítulos subsequentes:

- Capítulo 2: São apresentados os conceitos básicos para a compreensão da proposta desta pesquisa, embasando o princípio da Programação e Controle da Produção e seus problemas, juntamente com as formalizações das restrições, e métodos de avaliação dos resultados. Ao final são apresentados os algoritmos utilizados na hibridização da proposta.
- Capítulo 3: Este capítulo é responsável pela apresentação da fundamentação bibliográfica desta pesquisa, com trabalhos relacionados que baseiam o desenvolvimento da proposta. Uma análise geral dos trabalhos é exposta ao término.
- Capítulo 4: É apresentado a formulação do problema, o algoritmo proposto para a resolução do mesmo somado aos materiais e métodos que auxiliam este procedimento. Para complementar, todas as técnicas que compõem a proposta são explanadas para o entendimento do funcionamento da hibridização.
- Capítulo 5: São exibidos os resultados dos testes preliminares utilizando as combinações entre PSO + RRHC, PSO + SA e PSO + TS, propostas nesta pesquisa, finalizando com uma comparação dos testes com os trabalhos expostos no Capítulo 3 e uma comparação entre as meta-heurísticas.
- Capítulo 6: São apresentados a conclusão e trabalhos futuros factíveis

Capítulo 2

CONCEITOS BÁSICOS

2.1 Programação e Controle da Produção

O conceito de Programação e Controle da Produção pode ser explanado como o ato de designar e sequenciar operações em recursos produtivos determinados, da forma menos custosa possível mantendo a coesão com as restrições impostas. HART; ROSS; CORNE (2005), e PINEDO (2012) elucidam como técnica de destinar recursos limitados a determinadas tarefas por um intervalo de tempo, com finalidade em gerar saídas esperadas no prazo delineado, respeitando também restrições de tempo, precedência e de capacidade produtiva dos recursos (máquinas).

Há ainda uma gama de abordagens sobre a definição de programação da produção, HART; ROSS; CORNE (2005) os classificam, a partir da relação entre o número de recursos, a quantidade de ordens de produção (*jobs*) e operações, e os diferentes tipos de restrições impostas para este enredo. Os modelos de programação mais dissertados e analisados na prática, ainda segundo HART; ROSS; CORNE (2005) são:

- *Single Machine-shop Scheduling* (SMS): cada *job* possui apenas uma operação a ser executada por um único recurso produtivo;
- *Parallel Machines-shop Scheduling* (PMS): nesse tipo de problema, cada *job* pode ser operado por qualquer um dos m recursos disponíveis;
- *Flow-shop Scheduling* (FS): cenário em que existem m recursos e n *jobs*, sendo que cada *job* apresenta um conjunto de operações (estágios de

processamento) pré-ordenadas e as sequências de execução nos recursos disponíveis são as mesmas para todos os *jobs* envolvidos;

- *Job-shop Scheduling* (JS): tal forma de produção difere do anterior pelo fato de que, nesse caso, cada *job* apresenta uma sequência própria de execução de suas operações nos recursos produtivos disponíveis, podendo o recurso ser ou não compartilhado por outro *job* do sistema considerado; e
- *Open-shop Scheduling* (OS): este tipo de produção pode ser interpretado como um caso particular do JS, onde os *jobs* verificados não apresentam restrições internas de ordenamento de operações, ou seja, suas operações não possuem relações de precedência pré-estabelecidas.

Além destes modelos aludidos, existe o *Flexible Job-shop Scheduling* (FJS), que é uma generalização que respeita as premissas do modelo base (JS), porém possui complexidades complementares que resultam no FJSP, estas complexidades serão explicadas na próxima seção.

2.2 Definição de FJSP

PINEDO (2009) ilustra que problemas dessa ordem são caracterizados pela imposição de se processar n *jobs* utilizando m recursos produtivos, sendo que cada recurso está contido em um conjunto de recursos M que processa este determinado *job*. Conforme um *job* é designado à produção, este pode optar por um recurso produtivo dentre todos os disponíveis em seu conjunto de opções, respeitando as condições atuais do recurso, posto que este cenário aumenta significativamente o número de soluções factíveis. Na Tabela 1 consta o exemplo de um problema 3x3, ou seja, três *jobs* que serão processados em três recursos. Também é possível observar que cada *job* possui três operações, e que algumas operações não são processadas em alguns recursos, contendo o tempo de processamento igual a 0, caracterizando assim em um problema de job-shop parcialmente flexível (P-FJSP), este conceito será elucidado melhor nos capítulos 3 e 4.

Tabela 1: Exemplo de um problema 3x3.

job	$O_{i,j}$	M_1	M_2	M_3
1	$O_{1,1}$	2	0	3
	$O_{1,2}$	6	0	9
	$O_{1,3}$	8	7	7
2	$O_{2,1}$	3	2	6
	$O_{2,2}$	2	2	3
	$O_{2,3}$	5	2	4
3	$O_{3,1}$	2	0	1
	$O_{3,2}$	4	6	5
	$O_{3,3}$	0	1	3

O que denota um FJSP é a capacidade de pelo menos uma operação de todo o problema, ser processada por dois ou mais recursos, isto é possível pois os recursos deste tipo de problema possuem as mesmas aptidões, porém podem variar seus tempos de processamento ou não. Um exemplo dessa alternância pode ser visto na Figura 1, que é uma representação de um dos roteamentos possíveis usando arcos conjuntivos. A Figura 1 ainda indica que, conforme a Tabela 1 também apresentou, a operação $O_{1,3}$ pode ser processada pelos recurso M_1, M_2 e M_3, porém para este roteamento foi selecionado o M_3.

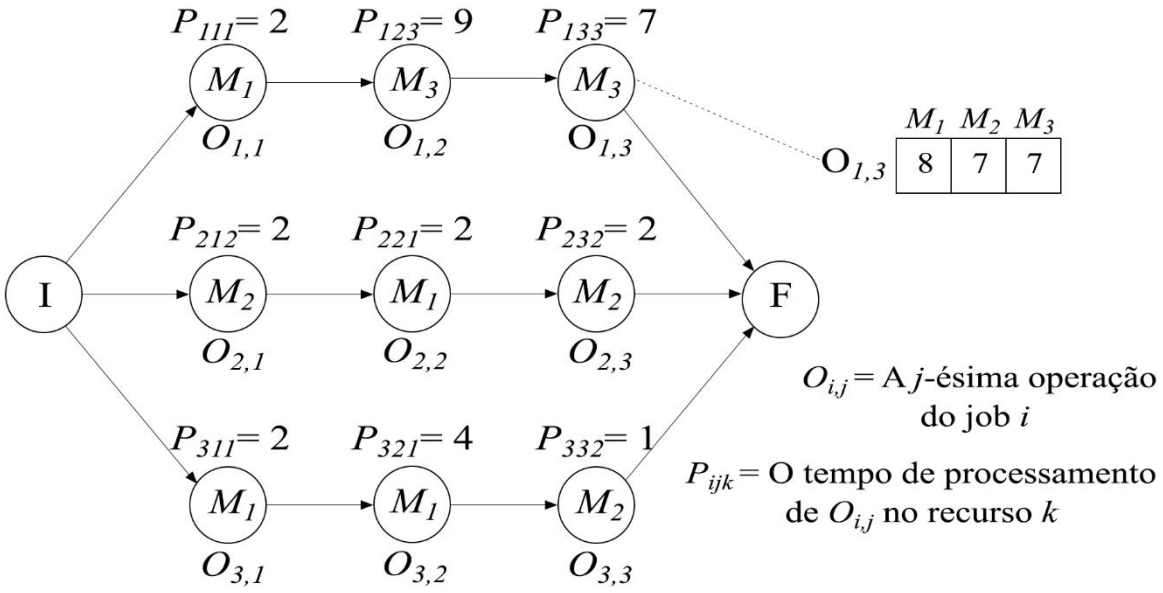

Figura 1: Possível roteamento do FJSP exemplificado na Tabela 1.

As abordagens de resolução do FJSP se distinguem pela decomposição ou não do problema em subproblemas menos complexos que podem ser atacados isoladamente em duas fases. Como supradito, A Figura 1 é um roteamento das operações, cujo procedimento consiste na atribuição de cada uma das operações a um dos recursos produtivos habilitados a realizar seu processamento, essas atribuições determinam os caminhos que as operações percorrerão através dos recursos. A segunda fase é a programação das operações nos recursos, o que equivale à encontrar a melhor ordem de processamento das operações em cada recurso, um exemplo desse procedimento pode ser visto na Figura 2, representado através de um Gráfico de Gantt do problema 3x3, usando os mesmos valores da Figura 1. Gerar soluções para as duas fases implica na resolução do FJSP. Essa estratégia de resolução é denominada abordagem hierárquica, entretanto, em contraste com essa estratégia existe a abordagem integrada, na qual se busca obter soluções para o FJSP tratando simultaneamente roteamento e programação (XIA; WU, 2005).

É importante ressaltar que na decomposição do problema em dois subproblemas, gerar o melhor roteamento não necessariamente implica na aquisição da melhor programação, e consequentemente não é adquirida a melhor aptidão, devido ao fato de que o algoritmo responsável pela programação tem característica de otimização, não garantindo assim a melhor solução.

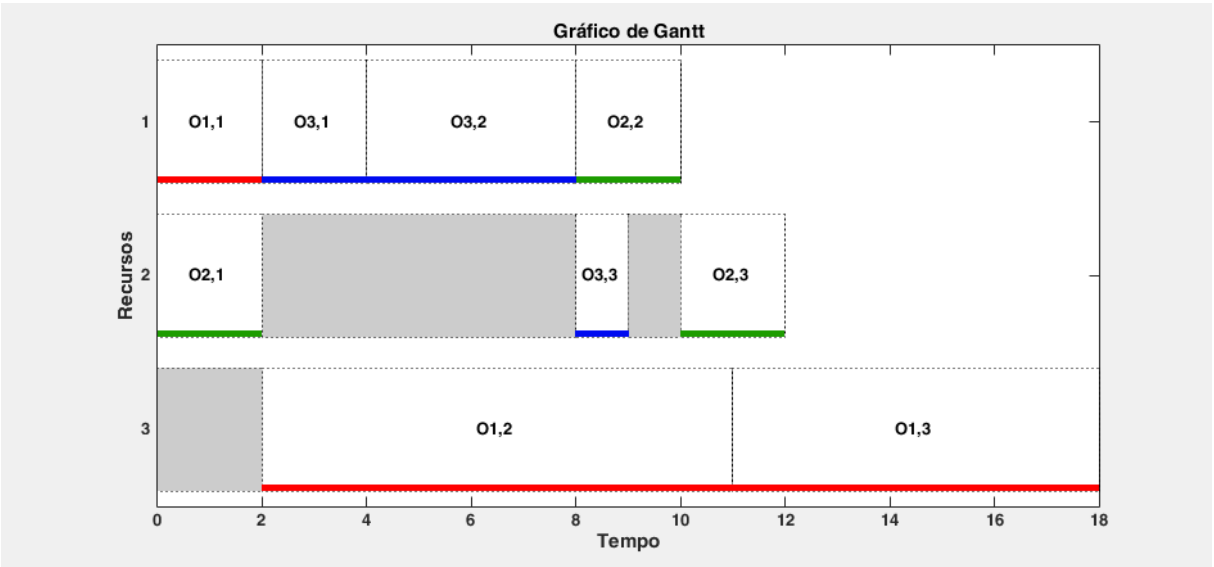

Figura 2: Gráfico de Gantt para o problema exemplificado.

2.3 Índice de desempenho

O índice de desempenho adotado varia de acordo com o objetivo do PCP, posto que a singularidade de um modelo de programação pode gerar um preceito de avaliação incomum, não obstante, HART; ROSS; CORNE (2005) mencionam os índices de desempenho que são usados comumente em trabalhos científicos, tendo em evidência o *makespan*, que é identificado como o tempo decorrido entre o início do primeiro e o fim do último *job* do conjunto a ser processado. Junto ao *makespan*, outros dois critérios foram escolhidos para serem minimizados, a diminuição do tempo total de um recurso crítico e o total de tempo gasto em todos os recursos. A formulação matemática do problema pode ser representada matematicamente da seguinte forma (ZHANG et al., 2009):

n: número de *jobs*;
m: número de recursos produtivos;
n_i: Número total de operações do *job* i;
$O_{i,j}$: A j-ésima operação do *job* i;
$M_{i,j}$: O conjunto de recursos disponíveis para a operação $O_{i,j}$;
P_{ijk}: O tempo de processamento de $O_{i,j}$ no recurso k;
t_{ijk}: Tempo de início da operação $O_{i,j}$ no recurso k;
$C_{i,j}$: O tempo de conclusão da operação $O_{i,j}$;
i, h: índices de *jobs*, i, h = 1, 2,..., n;2
k: índice de recursos, k = 1, 2,..., m;
j, g: índices de sequência de operação, j, g = 1, 2,..., ni;
C_k: O tempo de conclusão de M_k;
W_k: A carga de trabalho de M_k.

A função de minimização do FJSP é definida por:

Minimize $f(x) = (f_1(x), f_2(x),..., f_q(x),..., f_Q(x))$; (1)

$x \in \Omega$

Onde Ω é um espaço de solução factível, $x = \{x_1, x_2, ..., x_p, ..., x_P\}$ é o conjunto de de variáveis de decisão p-dimensional (inteira, contínua ou discreta) ($1 \leq p \leq P$), sendo uma delas a solução para função objetivo corrente; $f_q(x)$ é a q-ésima função objetivo ($1 \leq q \leq Q$). Em (1) é visado encontrar a minimização do tempo de todo os objetivos para no passo seguinte ponderá-los a partir de suas influências. Mais detalhes sobre as definições sobre os termos da otimização multiobjetivo podem ser vistos em DEB (2001). Prosseguindo com a notação tem-se:

$$\text{Min } f_1 = \max_{1 \leq k \leq m}(C_k) \tag{2}$$

$$\text{Min } f_2 = \max_{1 \leq k \leq m}(W_k) \tag{3}$$

$$\text{Min } f_3 = \sum_{k=1}^{m} W_k \tag{4}$$

$$C_{ij} - C_{i(j-1)} \geq C_{ijk} x_{ijk}, j = 2, \dots, n_i; \ \forall i, j \tag{5}$$

$$\left[\left(C_{hg} - C_{ij} - t_{hgk}\right) x_{hgk} x_{ijk} \geq 0\right] \vee \left[\left(C_{ij} - C_{hg} - t_{ijk}\right) x_{hgj} x_{ijk} \geq 0\right], \quad \forall (i,j), (h,g), k \tag{6}$$

$$\sum_{k \in M_{i,j}} x_{ijk} = 1, \forall i, j \tag{7}$$

As equações (2), (3) e (4) representam a obtenção dos mínimos das funções que objetivam a pesquisa; a inequação (5) garante o cumprimento da regra de precedência das operações; já a inequação (6) atesta que apenas uma operação pode ser processada por vez em um recurso produtivo; e a equação (7) assegura que os recursos sejam selecionados exclusivamente do conjunto de recursos disponíveis do *job* vigente.

2.4 Enxame de Partículas

Motivado pelo estudo do comportamento social de cardumes de peixes, algumas espécies de pássaros migratórios, e dos seres humanos, foi criado o algoritmo de otimização por enxame de partículas (*Particle Swarm Optimization* - PSO), desenvolvido por James Kennedy e Russel Elberhart em 1995, tem como princípio a minimização de funções de domínio contínuo (KENNEDY; EBERHART, 1995).

O PSO emergiu de experiências com algoritmos que modelam o "comportamento social" observado em muitas espécies de pássaros e cardumes de peixes, e até mesmo dos humanos. Duas concepções regem a atuação do algoritmo, uma é a capacidade de cada partícula (indivíduo de uma população) em quantificar a eficácia de sua própria experiência, chamada de aprendizado cognitivo, e a iterativa troca de experiências com seus vizinhos, rotulada como aprendizado social. Assim sendo, uma partícula percorre o espaço de solução, com velocidade de deslocamento consequente do aprendizado sociocognitivo. KENNEDY; EBERHART (1995) utilizaram três princípios para descrever o processo de aprendizado:

- Avaliar – os indivíduos possuem a capacidade de sentir o ambiente de forma a estimar seu próprio comportamento;
- Comparar – os indivíduos usam uns aos outros como referência comparativa;
- Imitar – a imitação é central em organizações sociais humanas e é importante para a aquisição e manutenção das habilidades mentais.

A capacidade de interação com outros indivíduos, estimativa de experiência própria e de imitação do comportamento, atribui ao algoritmo a aptidão de encontrar soluções globais, pois os indivíduos são inicializados aleatoriamente e ao trocar informações com seus vizinhos próximos ou serem influenciados pela melhor resposta, migram no espaço de busca em direção a esse comportamento, como a intensidade do deslocamento também é aleatória, é garantido o caráter global de otimização.

2.4.1 Algoritmo PSO

No PSO, as partículas se deslocam em um espaço de solução Rd, onde d é a dimensão do espaço. As variações nos atributos dessas partículas levam a novos posicionamentos de partículas, denominados pontos no espaço. Essas variações ocorrem por consequência da influência do comportamento sociocognitivo.

Os pontos de tendência para as partículas são definidos por duas métricas, a primeira é a melhor experiência global (*gB*), experiência esta que está conectada a todos os membros de uma população, logo, o comportamento de cada partícula é induzido a inclinar-se para o ponto da melhor experiência global. A segunda métrica (*pB*) cria uma vizinhança para cada indivíduo composta por ele próprio e seus vizinhos mais próximos. Ambas as métricas são medidas por uma função de avaliação ($f(p)$), também chamada função objetivo ou de aptidão (*fitness*), que corresponde à otimalidade da solução do problema.

A nova posição de uma partícula (p_i) é gerada a partir de sua posição atual ($x_{i,t}$), aplicada uma velocidade ($v_{i,t+1}$) e as respectivas influências, local (*pB*) e global (*gB*). Considerando este enredo, a premissa do PSO é estabelecida por meio do deslocamento de suas partículas, que, adquirindo velocidades adaptativas e acessando suas experiências junto as dos vizinhos (melhor posição local) e a do grupo todo (melhor posição global), movimenta-se ao longo do espaço de solução, procurando pelo ótimo global. Deste modo sempre haverá um indivíduo que detém a melhor solução, a qual influenciará todos os outros (*gB*) e os melhores vizinhos que servem de base para os indivíduos próximos (*pB*), como visto na Figura 3.

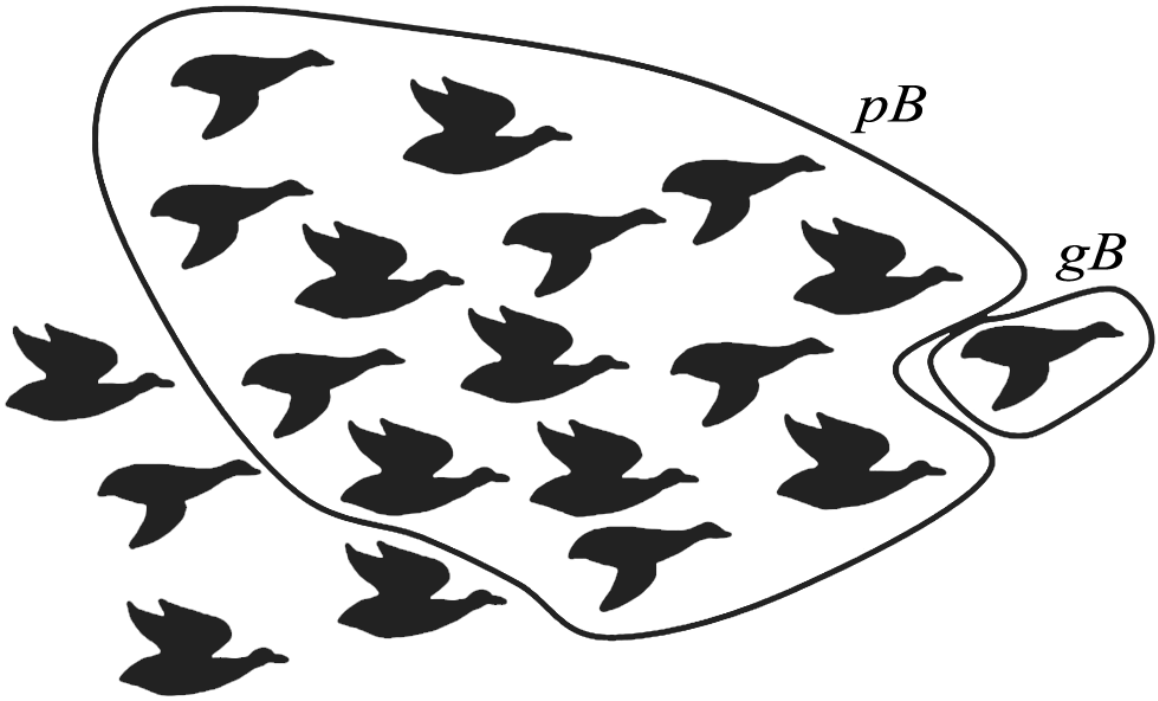

Figura 3: Comportamento migratório de um bando de pássaros e sua relação com pB e gB
Fonte: Adaptado de ZUBEN; ATTUX (2009).

A velocidade da partícula é calculada como mostra a equação (8).

$$v_{i,t+1} = v_{i,t} + c_1 * (pB - x_{i,t}) + c_2 * (gB - x_{i,t}) \quad (8)$$

Onde c_1 e c_2 são constantes inseridas, que representam as diretrizes do comportamento “cognitivo” e “social”.

Assim que a velocidade da partícula é encontrada, a posição da partícula i na próxima iteração é atualizada sendo, o valor da posição anterior mais a velocidade de deslocamento calculada, podendo ser representada como visto na equação (9).

$$x_{i,t+1} = x_{i,t} + v_{i,t+1} \quad (9)$$

Para que a velocidade de uma partícula não gere partículas com soluções não contidas no espaço de busca do problema corrente, são impostos limites máximos e

mínimos ($max(x)\ e\ min(x)$) para seus valores em cada dimensão (d) do espaço de busca, com a seguinte condição:

Se x_i > max(x) então x_i = max(x),
Senão se x_i < min(x) então x_i = min(x).

O algoritmo PSO é repetido até que um critério de parada seja alcançado ou as mudanças nas velocidades das partículas estejam perto de zero (estagnação). O algoritmo, em sua forma original, é descrito no Pseudocódigo 1.

Pseudocódigo 1: Algoritmo Enxame de Partículas

1. Determine o número de partículas P da população.
2. Inicialize aleatoriamente a posição inicial (x) de cada partícula p de P.
3. Atribua uma velocidade inicial (v) igual para todas as partículas.
4. Para cada partícula p em P faça:
   (a) Calcule sua aptidão $f_p = f(p)$.
   (b) Calcule a melhor posição da partícula p até o momento (pB).
5. Descubra a partícula com a melhor aptidão de toda a população (gB).
6. Para cada partícula p em P faça:
   (a) Atualize a velocidade da partícula pela equação:
   $v_{i,t+1} = v_{i,t} + c1 * (pB - x_{i,t}) + c2 * (gB - x_{i,t})$
   (b) Atualize a posição da particular pela equação:
   $x_{i,t+1} = x_{i,t} + v_{i,t+1}$.
7. Se condição de término não for alcançada, retorne ao passo 4.

2.4.2 Avanços do algoritmo PSO

Um dos principais problemas do PSO desde sua criação é a fácil convergência prematura em um ótimo local, por conta de sua influência do melhor resultado global, e do melhor resultado da vizinhança de uma partícula. Para melhorar este quesito somado a uma melhor precisão, foram desenvolvidos trabalhos com alterações nos seguintes componentes: peso de inércia da velocidade (w) (SHI; EBERHART, 1998); fator de constrição (x), que é um fator de amortecimento baseado nos parâmetros

cognitivo (c_1) e social (c_2) para limitar a velocidade da partícula (EBERHART; SHI, 2000); operação de cruzamento baseado nos mecanismos de reprodução dos algoritmos genéticos (LØVBJERG; RASMUSSEN, 2001); o uso de distribuição gaussiana nos parâmetros de inércia (w), cognitivo (c_1) e social (c_2) (MIRANDA; FONSECA, 2002); autoadaptação (LU; HOU; DU, 2006); inclusão de um operador de evolução diferencial (DEPSO) na partícula do esquema encontrado (WEN-JUN ZHANG; XIAO-FENG XIE, 2003); filtro de Kalman (KPSO) para atualização das posições das partículas (MONSON; SEPPI, 2004); CSV-PSO (CHEN; FENG, 2005), que ajusta dinamicamente e não-linearmente o peso de inércia, o limite da velocidade de voo e o espaço de voo das partículas; vizinhança independente com subenxames independentes (INPSO) (GROSAN et al., 2005); turbulência (TPSO) e turbulência adaptada nebulosamente (FATPSO) (HONGBO LIU; ABRAHAM, 2005); PSO acelerado (APSO) (YANG, 2010); Caos melhorado (CAPSO) (GANDOMI et al., 2013).

Além da alteração da base do PSO, existem pesquisas onde o PSO é proposto como um fator de aprimoramento para outra meta-heurística, somando para a otimização de um objetivo ou mais objetivos, esse conceito é o de hibridização. Algumas das meta-heurísticas usadas são: algoritmos genéticos (NIE RU; YUE JIANHUA, 2008); redes neurais (BASHIR; EL-HAWARY, 2009); sistemas nebulosos (JUANG; WANG, 2009); evolução diferencial (WICKRAMASINGHE; LI, 2008), e *clustering* (PEI; HUA; HAN, 2008). Outros exemplos de hibridizações, como Busca Tabu (TS) e Arrefecimento Simulado (SA), serão descritos no capítulo 3, por serem base deste trabalho.

2.5 Arrefecimento Simulado

A utilização do algoritmo para resolução de problema de otimização combinatória tem gerado bons resultados e consequentemente novos estudos devido a sua simplicidade e aplicabilidade em diversas áreas científicas como engenharia da produção, matemática, física e ciência da computação.

Criado por METROPOLIS et al. (1953) e introduzido à esfera de problemas de otimização combinatória por KIRKPATRICK; GELATT; VECCHI (1983) e ČERNÝ (1985), simula conceitualmente o processo de arrefecimento e cristalização de corpos

sólidos, onde um corpo sólido é aquecido previamente com a finalidade de aquisição de livre movimentação das partículas pela quebra das ligações intermoleculares, que só é possível com a aquisição de energia deste aquecimento. Esta movimentação tende a ser aleatória, gerando uma reorganização de forma reticular e singular proporcionalmente a diminuição da temperatura e solidificação do corpo. Como o arrefecimento prima por ser feito de forma lenta, propícia a estabilização de uma estrutura equilibrada e livre de defeitos por ser gerada com energia mínima, tendo como estágio final a obtenção de uma estrutura cristalina de energia e com interação mínima entre as partículas, ou seja, o ponto mínimo local.

2.5.1 Algoritmo SA

O algoritmo é iniciado analogamente ao processo de arrefecimento em uma determinada temperatura, chamada de temperatura inicial (T_0), a cada nova iteração uma nova solução S' é gerada derivada da vizinhança da solução atual S, esta nova solução é submetida à função objetivo que rege o algoritmo $f(S')$, concebendo assim uma variação como mostra a equação (10).

$$\Delta = f(S') - f(S) \tag{10}$$

Se a variação for negativa (houve melhora), a atribuição da nova solução é aceita, caso contrário, a transição para a nova solução é aceita dada uma probabilidade:

Se $\Delta < 0$ então $S = S'$
Senão Se $p \in [0,1] \leq exp(-\Delta/T)$ então $S = S'$

Onde T é a variável que representa a temperatura, desta forma a probabilidade de um estado com energia superior ser aceito diminui proporcionalmente à temperatura. O algoritmo SA é geralmente inicializado com um valor alto para a temperatura, e então este valor é gradativamente diminuído através de um coeficiente de resfriamento (α), como visto na equação (11).

$$T = T\ x\ \alpha \tag{11}$$

A cada temperatura, a busca por uma solução nova que satisfaça a diminuição de energia é feita por uma quantidade determinada de iterações. Quando T atingir a temperatura final T_{end}, o algoritmo é encerrado, como mostra o Pseudocódigo 2.

Pseudocódigo 2: Algoritmo Arrefecimento Simulado

1. Define a quantidade de iterações por temperatura e a temperatura final T_{end}
2. Gera uma solução inicial (S).
3. Avalia a aptidão de S, chamando-a de $f(S)$.
4. Define uma temperatura inicial (T_0).
5. De 1 até a quantidade definida de iterações por temperatura faça:
   (a) S' = gera_vizinho(S) – gerar uma nova solução a partir de S.
   (b) Se $f(S') \leq f(S)$ então
   $S = S'$.
   Senão
   $\Delta = f(S') - f(S)$.
   Se $p \in [0,1] \leq exp(-\Delta / T)$ então
   $S = S'$.
6. Atualize a temperatura ($T = T\ x\ \alpha$).
7. Se T_{end} não for alcançada, retorne ao passo 5.

2.6 Busca Tabu

O conceito de Busca Tabu (TS) foi criado por GLOVER (1989), utiliza de mecanismos de busca local para solucionar problemas de otimização, e tem obtido resultados satisfatórios em problemas combinatoriais e de programação (ZHANG et al., 2009). O TS permite que soluções ruins sejam aceitas se uma melhor não for encontrada, soluções que já foram aceitas são designadas a uma lista de soluções proibidas, chamada de lista tabu, prevenindo assim que o algoritmo considere essas soluções novamente. O algoritmo consiste de componentes que governam sua operação, como o tipo de memória, tamanho da lista tabu, critério de parada e movimento das soluções (função de vizinhança).

Os tipos de memória do algoritmo podem ser divididos em três categorias (GLOVER, 1990):

- Período curto, é um tipo de lista com soluções recentemente usadas. Se uma solução for gerada que pertença a essa lista, ela só poderá ser usada após um tempo de expiração.
- Período intermediário, é um tipo de lista elitista, contém boas soluções encontradas e tem como objetivo intensificar a buscar de soluções próximas, reiniciando a busca a partir das soluções contidas na lista. O processo perdura por um determinado número de iterações ou até a nenhuma nova solução for encontrada.
- Período longo, é o tipo mais complexo de lista, tem como objetivo a diversificação, ou seja, procurar por soluções inexploradas. Na maioria dos casos mantém uma base de atributos de soluções boas e comuta com atributos ainda não utilizados resultando assim soluções atípicas. O processo é repetido até que uma determinada condição de parada seja satisfeita.

Os outros componentes que governam o algoritmo são semelhantes a outros tipos de meta-heurísticas e serão explanados na descrição do algoritmo.

2.6.1 Algoritmo TS

No contexto de FJSP o algoritmo TS pode ser caracterizado como: Uma dupla (v, k) como elemento tabu, v é a operação a ser movida e k o recurso onde v será movida. $Tlist$ é a lista tabu, uma matriz que guardará todas as soluções tabus. O parâmetro s é interpretado como uma solução de programação do FJSP, $V(s)$ é a sua vizinhança, $sBestNeigh$ é o melhor vizinho de $V(s)$, e $F(s)$ é o valor gerado pela função de aptidão de s. Para não gerar uma imensa quantidade de possibilidades no ato de gerar vizinhos, uma melhoria para a função de vizinhança deve ser adotada, nesta pesquisa foi utilizado o conceito de caminho crítico que será explicado no Capítulo 4, uma vez que somente vizinhos que estão no caminho crítico são considerados. O pseudocódigo pode ser analisado abaixo no Pseudocódigo 3.

Pseudocódigo 3: Algoritmo Busca Tabu

1. Define tamanho da lista tabu e quantidade iterações do TS.
2. Gera uma solução inicial (s).
3. Avalia a aptidão de s, chamando-a de $F(s)$.
4. De 1 até a quantidade de vizinhos de $V(s)$ faça:
   (a) s' = gera_vizinho(s) – gerar uma nova solução a partir de s, atribuindo para v a operação que foi movida e para k o recurso dessa operação.
   (b) Avalia $F(s')$.
   (c) Se $F(s') < sBestNeigh$ então
   $$sBestNeigh = (v, k)$$
5. Atualiza $Tlist$ com $sBestNeigh$
6. Se $F(sBestNeigh) < F(s)$ então
   $$s = sBestNeigh$$
7. Se a quantidade de iteração do TS não for alcançada, retorne ao passo 4.

2.7 Reinício Aleatório de Subida de Colina

A Subida de Colina (*Hill Climbing* - HC) é mais um método da família de buscas locais, uma técnica matemática de otimização relativamente fácil de implementar (MOTAGHEDI-LARIJANI; SABRI-LAHAIE; HEYDARI, 2010). O algoritmo é classificado como guloso (*greedy*), porque depois de cada iteração do método, ele só aceitará uma nova solução se essa gerar uma melhora, resultando em grandes possibilidades de estagnação em ótimos locais. O procedimento habitual é de iniciar com uma solução aleatória, que na maioria dos casos é ruim, e iterativamente gerar pequenas alterações a fim de adquirir melhores resultados. Quando o algoritmo atingir o critério de parada, ele termina.

Para aprimorar a eficiência do algoritmo, algumas variações foram propostas, como é o caso da Subida de Colina Mais Íngreme (*Steepest Ascent Hill Climbing*), uma técnica que compara todos os possíveis movimentos de uma vizinhança atual, para seguir o melhor resultado, porém falha como o HC tradicional, se não houver um melhor resultado dos movimentos.

A Subida de Colina Estocástica difere do original pelo fato de aceitar resultados piores, este procedimento aceita uma solução de forma aleatória, baseada na variação do resultado e da iteração corrente, quanto maior for a variação negativa e a iteração, menor a chance de aceitação desta solução.

Outra variante do HC é o Reinício Aleatório de Subida de Colina (RRHC), o qual de acordo com RUSSELL; NORVIG (1995), é um conjunto de buscas de HC de pontos iniciais aleatórios, sendo executadas cada uma até um critério de parada ou não obtiver mais melhora. O RRHC é usado nesta pesquisa, porém iniciando de uma solução estocástica que será elucidada no Capítulo 4, e então, após um número de iteração sem melhora, o algoritmo salva o estado atual e reinicia de uma solução estocástica novamente.

2.7.1 Algoritmo RRHC

Este algoritmo é tecnicamente simples de ser ilustrado, por depender de poucas variáveis. As únicas variáveis que auxiliam a operação do RRHC é a quantidade de iterações e a quantidade de iterações sem melhora $n\_restart$, todavia como os algoritmos de busca local, é regido por uma função de aptidão $f(s)$, sendo s uma solução factível. Para melhor compreensão vide Pseudocódigo 4.

Pseudocódigo 4: Algoritmo Reinício Aleatório de Subida de Colina

1. Define a quantidade de iterações do HC, $n\_restart$, e a solução inicial s.
2. Avalia a aptidão de s, chamando-a de $f(s)$.
3. $iter\_restart = 0$.
4. Se $iter\_restart > n\_restart$ então
   s recebe uma solução inicial.
   $iter\_restart = 0$
5. s' = gera_vizinho(s) – gerar uma nova solução a partir de s.
6. Se $f(s') \leq f(sBest)$ então
   $s = s'$.
   $sBest = s'$.
   Senão
   $iter\_restart = iter\_restart + 1$.
7. Se a quantidade de iterações não for atingida, retorne ao passo 4.

2.8 Tendência na Resolução do FJSP

Como visto no item 2.2, o FJSP é usualmente dividido em dois subproblemas, o de roteamento e o de programação, essa divisão é adequada pelo fato de que cada um apresenta características singulares que favorecem a aplicação de algoritmos de busca local e global. Essa diferenciação se dá porque no subproblema de roteamento é mais cabível explorar de forma ampla o espaço de busca, tendo como base soluções anteriores para possível direcionamento, enquanto o subproblema de programação varia sua vizinhança de forma simples, na maioria das vezes só aceitando uma nova solução se houver melhora e com grande probabilidade de ficar preso em ótimos locais. A distinção entre os algoritmos de busca local e global é explanada de forma empírica no estudo de SCHONLAU; WELCH; JONES (1998).

O algoritmo PSO como ilustrado acima, possui alta performance na otimização de problemas de busca global, como é o caso do roteamento, uma vez que sua parametrização pode favorecer a exploração de um amplo espaço de busca, à medida que os algoritmos SA, TS e RRHC são apropriados para o subproblema de programação, posto que são algoritmos de otimização que alteram suas soluções de forma simples, com pouca variância em sua vizinhança. A interação entre os algoritmos é comumente feita através da codificação de uma solução com um possível roteamento no subproblema de roteamento, interpretada e avaliada no subproblema de programação, para que seu resultado sirva de base para as futuras gerações do algoritmo de busca global no subproblema de roteamento. Este procedimento será melhor descrito no capítulo 4 e ilustrado por um fluxograma.

É notório que a utilização de paralelismo na resolução do FJSP aplicando o PSO é conveniente, visto que é um algoritmo baseado em população, e cada partícula é somente influenciada por seu melhor local e global na geração corrente. Assim é possível avaliar em paralelo cada solução de uma população em uma determinada geração. A codificação do algoritmo em paralelo permite que seja explorado o potencial das máquinas múltiplo que gozam de múltiplos núcleos, resultando assim em um processamento mais rápido do problema.

Capítulo 3

REVISÃO BIBLIOGRÁFICA

3.1 Trabalhos Relacionados

Desde seu primeiro trabalho divulgado sobre a resolução do FJSP, o tema vem sido divagado e estudado até os dias atuais, tendo aplicações de novos algoritmos, inserções de restrições, abordagens, entre outros. Dado que o FJSP pode ser considerado mais próximo de uma produção real de fabricação, e é mais complexo que o JSP tradicional, tornando-o mais propenso à estudos.

3.2 Regras de Despacho e Busca Tabu

O precursor dos trabalhos sobre FJSP usando a decomposição em dois subproblemas foi (BRANDIMARTE, 1993), sua decomposição visava uma aproximação da resolução do FJSP. Primeiro foi proposta a solução do subproblema de roteamento com o auxílio de regras de despacho, para depois resolver o subproblema de programação, em ambos os subproblemas foram utilizados a heurística de Busca Tabu (TS). As regras de despacho são rotuladas como uma estratégia de programação distribuída, a qual cada regra recebe uma prioridade para o processamento de uma determinada tarefa referente às suas condições. A regra com maior prioridade é escolhida para o processamento do *job* corrente.

Com a abordagem das regras de despacho é possível codificar roteiros de produção baseados em diversas funções objetivos, visto que as regras podem ser representadas por um ou mais recursos produtivos aptos a processar um determinado *job*, respeitando as restrições formuladas nos números (5), (6) e (7) do capítulo 2.3. As regras mais comumente vistas em ambiente de produção são:

1. O Tempo de Processamento Mais Curto (Shortest Processing Time SPT), tendo P_{ij} como o tempo de processamento do recurso m na operação ij, a prioridade desta regra é dada por:

$$\frac{1}{P_{ij}m} \tag{12}$$

   Quando um peso w é dado a um job a regra SPT passa a ser ponderada (WSPT), resultando na seguinte formulação:

$$\frac{w_i}{P_{ij}m} \tag{13}$$

2. Regra de Menor Quantidade de Trabalho Restante (Least Work Remaining LWKR), que atribui sua prioridade conforme abaixo:

$$\frac{1}{\sum_{q \in A_{ij}} P_{iq}} \tag{14}$$

   Onde A_{ij} é o conjunto de operações restantes no plano de processo do *job* J_i e

$$P_{iq} = \frac{1}{\varepsilon_{iq}} \sum_{1 \in \varepsilon_{iq}} P_{igm} \tag{15}$$

   é a média do tempo de processamento da operação O_{iq}

3. Maior Quantidade de Trabalho Restante (Most Work Remaining MWKR), que rege sua prioridade pelo cálculo:

$$\sum_{q \in A_{ij}} P_{iq} \tag{16}$$

O algoritmo TS permite que sua busca explore soluções adicionais mesmo que o melhor valor de função objetivo não seja superado, posto que essas soluções não sejam proibidas. As novas soluções são derivadas com base nas últimas soluções geradas, essa transformação é denominada de tabu e perdura para as próximas T iterações, cujo T é a quantidade de iterações para uma transformação tabu (movimento pelo espaço de busca). Uma solução é considerada proibida quando é obtida aplicando a transformação tabu na solução corrente.

A partir da definição da melhor regra, que neste caso foi a MWKR, foi elaborado um plano para que o algoritmo TS pudesse operar em duas vias na interação dos subproblemas, sendo uma para o processo normal de criação de rotas e outro para a consideração de caminhos de operações críticas, como por exemplo, rotas que resultam em tempos de processamento muito longos, dando um critério de *feedback* para o algoritmo, visto na Figura 4.

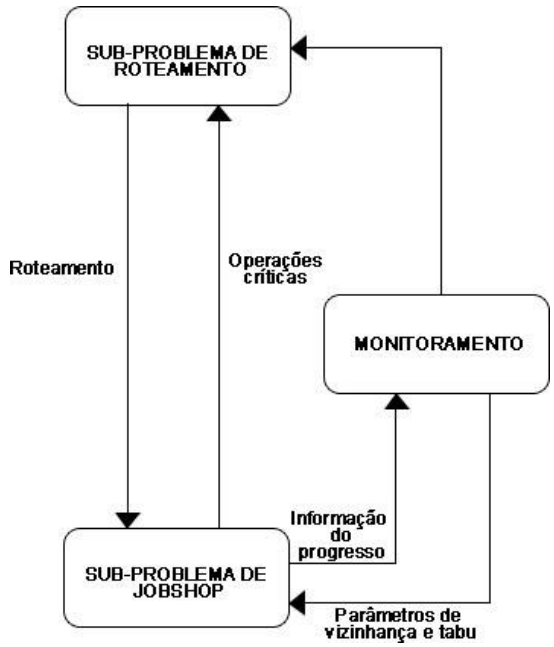

Figura 4: Fluxograma do algoritmo para resolução do FJSP por TS
Fonte: Adaptado de Brandimarte (1993).

A obtenção do caminho crítico é fundamental para a resolução de ambos os problemas, pois diminui consideravelmente a quantidade de vizinhos de uma solução, uma vez que somente operações contidas no caminho crítico são selecionadas para a transformação tabu, tanto para o problema de roteamento quanto o de *job-shop* (atualmente conhecido como programação). A execução do TS está amparada pelo processo de monitoramento, onde são definidos os parâmetros do algoritmo, estrutura da vizinhança e a avaliação do progresso para cada um dos subproblemas, similar a outras heurísticas de otimização (BRANDIMARTE, 1993).

No ato da publicação da pesquisa de Brandimarte ainda não havido sido criada a base de dados adotada para a comparação de performance, a Kacem data (KACEM; HAMMADI; BORNE, 2002), porém foi constatado através do *benchmark* criado pelo próprio Brandimarte, nominado como BR data, que a solução utilizando caminho de operações críticas para geração de vizinhança do algoritmo TS, somado as regras de despacho, são uma alternativa satisfatória para a resolução do FJSP.

3.3 FJSP Multiobjetivo usando Aproximação por localização

KACEM; HAMMADI; BORNE (2002), apresentaram um Algoritmo Genético (GA) gerado a partir do conceito de Aproximação de Localização (AL) para FJSP multiobjetivo. Para a validação da pesquisa dois tipos de codificações são propostos, o totalmente flexível e o parcialmente flexível.

Uma codificação é totalmente flexível, se todo recurso produtivo de um conjunto de recursos pode processar todas as operações de todos os *jobs*, respeitando também as fórmulas do capítulo 2.3 de precedência e processamento unitário por vez. Já para ser designada codificação parcialmente flexível, pelo menos um *job* do conjunto dos *jobs* não é processado por um determinado recurso em um conjunto de recursos produtivos, como visto na Tabela 2, onde a ausência de tempo que deveria ser nulo é substituído por um valor muito alto, neste caso o valor 999, por motivo de facilidade na codificação do algoritmo.

Tabela 2: Codificação onde tempos 999 são utilizados para denotar recursos que não processam determinados jobs.

Fonte: Adaptado de (KACEM; HAMMADI; BORNE, 2002).

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
1	$O_{1,1}$	5	3	5	3	3	999	10	9
	$O_{1,2}$	10	999	5	8	3	9	9	6
	$O_{1,3}$	999	10	999	5	6	2	4	5
2	$O_{2,1}$	5	7	3	9	8	999	9	999
	$O_{2,2}$	999	8	5	2	6	7	10	9
	$O_{2,3}$	999	10	999	5	6	4	1	7
	$O_{2,4}$	10	8	9	6	4	7	999	999
3	$O_{3,1}$	10	999	999	7	6	5	2	4
	$O_{3,2}$	999	10	6	4	8	9	10	999
	$O_{3,3}$	1	4	5	6	999	10	999	7
4	$O_{4,1}$	3	1	6	5	9	7	8	4
	$O_{4,2}$	12	11	7	8	10	5	6	9
	$O_{4,3}$	4	6	2	10	3	9	5	7
5	$O_{5,1}$	3	6	7	8	9	999	10	999
	$O_{5,2}$	10	999	7	4	9	8	6	999
	$O_{5,3}$	999	9	8	7	4	2	7	999
	$O_{5,4}$	11	9	999	6	7	5	3	6
6	$O_{6,1}$	6	7	1	4	6	9	999	10
	$O_{6,2}$	11	999	9	9	9	7	6	4
	$O_{6,3}$	10	5	9	10	11	999	10	999
7	$O_{7,1}$	5	4	2	6	7	999	10	999
	$O_{7,2}$	999	9	999	9	11	9	10	5
	$O_{7,3}$	999	8	9	3	8	6	999	10
8	$O_{8,1}$	2	8	5	9	999	4	999	10
	$O_{8,2}$	7	4	7	8	9	999	10	999
	$O_{8,3}$	9	9	999	8	5	6	7	1
	$O_{8,4}$	9	999	3	7	1	5	8	999

Neste contexto, primeiramente foi proposto a elaboração de um algoritmo AL para atribuição de *jobs* em recursos produtivos a fim de resolver o FJSP, focado no objetivo de redução da carga máxima de trabalho de todas os recursos e consequentemente o *makespan*, utilizando a regra SPT. Como o algoritmo AL é computacionalmente mais viável do que o GA, em primeira instância para a resolução de problemas menos complexos, o resultado gerado é melhor. Para o entendimento do funcionamento do algoritmo, o próprio autor do trabalho elaborou um pseudocódigo, mostrado no Pseudocódigo 5.

Pseudocódigo 5: Algoritmo de Aproximação de Localizção para resolução do FJSP

- Começo do Algoritmo de Programação
  - Inicializa o vetor de máquinas disponíveis $Dispo\_Machine[k] = 0$ para cada máquina M_k $(k \leq M)$;
  - Inicializa o vetor de *jobs* disponíveis $Dispo\_Job[j] = 0$ para cada *job* j $(j \leq N)$;
- Para $(i = 1, i \leq Max_j(n_j))$
  - Construa o conjunto E_i de operações para sequenciar de S: $E_i = \{\frac{O_{i,j}}{S_{i,j,k}} = 1, 1 \leq j \leq N\}$;
  - Classifique as operações de E_i de acordo com a regra de prioridade escolhida;
  - Para $(j = 1;\ 1 \leq j \leq N)$
    - Calcule os tempos iniciais seguindo a mesma ordem dada na classificação de Ei, de acordo com a fórmula: $t_{i,j} = Max(Dispo\_Machine[k], Dispo\_Job[j])$ tal que $S_{i,j,k} = 1$;
    - Atualize o vetor de máquinas disponíveis: $Dispo\_Machine[k] = t_{i,j} + d_{i,j,k}$;
      Atualize o vetor de jobs disponíveis: $Dispo\_Job[j] = t_{i,j} + d_{i,j,k}$;
  - Fim Para
- Fim Para
- Fim do Algoritmo de Programação.

Ainda na continuidade do trabalho com o intuito de solucionar problemas mais complexos foi proposto a utilização do modelo mostrado acima como desenvolvedor de esquemas e população inicial para o GA. O algoritmo genético permite que através de uma população inicial (soluções) representada por cromossomos haja evoluções que concebem novos cromossomos com a capacidade de minimizar funções em um espaço de busca global. O procedimento é análogo ao evolucionismo e possui métodos de *crossover*, mutação e seleção para a aquisição da solução ótima. O que difere do convencional é que o GA conta com duas formas de evolução, uma advinda

do esquema direto criado pelo AL e outra de uma população inicial, ambos são avaliados e validados. Esse procedimento é chamado de Algoritmo Genético Controlado (CGA), para melhor entendimento o fluxograma da hibridização consta na Figura 5.

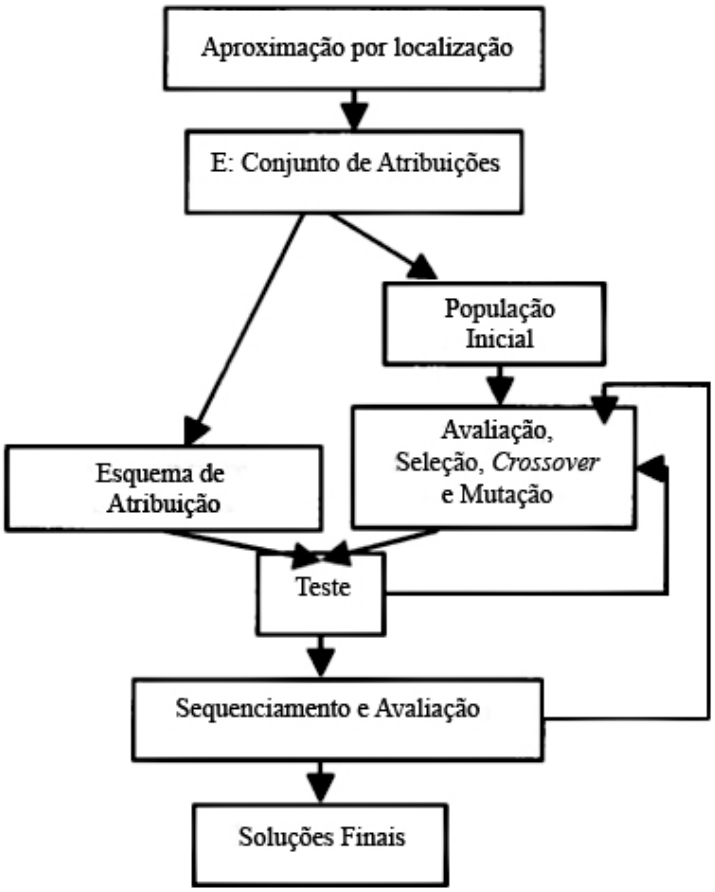

Figura 5: Fluxograma do algoritmo CGA
Fonte: Adaptado de (KACEM; HAMMADI; BORNE, 2002).

Ao ser comparado com um trabalho de mesmo escopo utilizando decomposição temporal, o algoritmo AL e GA obtiveram resultados 56% melhores acerca do volume total de trabalho de todos os recursos e do *makespan* (KACEM; HAMMADI; BORNE, 2002).

3.4 FJSP por PSO e SA

Uma pesquisa de referência sobre resolução de FJSP por Enxame de Partículas híbrido até o presente momento sobre algoritmo e que também é uma das bases de comparação deste objeto de trabalho é a de XIA; WU (2005), onde a hibridização proposta está delegada ao PSO a função de resolução do subproblema de roteamento e ao SA o subproblema de programação. O trabalho possui caráter multiobjetivo, e estes são transformados em um só através da soma de seus pesos, esta pesquisa trata de cenários totalmente flexíveis e parcialmente flexíveis. As particularidades deste modelo estão no mapeamento do problema e na geração da solução.

Os autores atribuem níveis de precedência para os recursos produtivos que são SPT de um determinado *job*, assim é gerada a população inicial de forma estocástica eliminando os recursos produtivos que tiverem o pior nível. Outro quesito importante do algoritmo está presente no caráter dinâmico da variável de peso para a dispersão da velocidade no PSO, devido o fator de que no começo do algoritmo o peso está alto, a procura de soluções é induzida a ser feita de uma forma mais dispersa no espaço de busca, e no fim estando baixo, o motiva a buscar por soluções locais, como denotado pela equação (17).

$$w = \max(w) - \left(\max(w) - \frac{\min(w)}{\max(iter)}\right) * iter \qquad (17)$$

Onde $\max(w)$ é o valor inicial do peso, $\min(w)$ o valor final, $\max(iter)$ o numero de iterações do algoritmo e $iter$ a iteração atual.

Deve-se frisar que a estratégia adotada para o cenário parcialmente flexível é a mesma de KACEM; HAMMADI; BORNE (2002), são atribuídos para os recursos produtivos que não processam determinadas operações, um valor muito alto para que sejam desconsiderados facilmente. Também é necessário citar que ao atualizar a velocidade das partículas no cenário parcialmente flexível, eventualmente um recurso produtivo que não processa uma determinada operação irá ser selecionado. Mais sobre este problema poderá ser visto no Capítulo 4.

Com intenção de avaliar a aptidão das soluções geradas, três métricas foram adotadas para serem os objetivos: (1) O *makespan*; (2) Tempo de trabalho de um recurso crítico; (3) O tempo total de trabalho de todos os recursos produtivos. Estes objetivos são ponderados no passo de avaliação do *fitness* do algoritmo.

No algoritmo SA, a escolha do método de geração de vizinhança tem uma grande influência na performance e tempo de processamento do algoritmo. Enquanto usar uma vizinhança vasta pode aumentar significativamente as chances de se obter a solução ótima, em contrapartida pode resultar em um custo computacional alto, para minimizar este problema ao contrário de gerar randomicamente uma nova solução, o algoritmo SA faz uma troca de pares entre seus elementos a fim de minimizar o tempo de espera. Assumindo p operações atribuídas em um recurso produtivo:

$$(1 \leftrightarrow 2), (2 \leftrightarrow 3), (3 \leftrightarrow 4), \dots, (p-1 \leftrightarrow p)$$

O fluxo iterativo do algoritmo é definido pela geração das soluções pelo PSO, e mapeamento pelo SA, finalizando com a validação e melhoria das soluções pelo PSO. Este procedimento é comparado com a pesquisa citada no item anterior de KACEM; HAMMADI; BORNE (2002), cuja melhora em um cenário de 56 operações chega ser de 50%.

3.5 PSO e TS

No mesmo segmento de PSO híbrido referenciado no item acima está o trabalho de ZHANG et al. (2009), com algumas peculiaridades, uma delas é que no subproblema de roteamento o algoritmo auxiliar é o de Busca Tabu, e a outra diferença foi a escolha do método de atualização da velocidade das partículas do PSO, um *crossover* semelhante ao procedimento do algoritmo genético.

Novamente é utilizado o caráter multiobjetivo com as mesmas funções a se minimizar, os mesmos cenários e restrições do trabalho anterior. Porém os autores propuseram uma melhoria ao PSO motivados pela probabilidade de estagnação do algoritmo por conta de seu comportamento. Almejando essa melhoria, uma adaptação foi feita na atualização da velocidade do algoritmo PSO, como visto na seção 2.4.1 o

algoritmo tende a gerar novas soluções baseadas no comportamento local ou global de suas partículas (sociocognitivo). Para suprir parte desse viés foi proposto um método de crossover para permutação das partículas representadas cromossomicamente (chamadas de *A-string*) e para permutação das operações das partículas (chamadas de *B-string*). Na permutação da configuração *A-string*, duas posições são selecionadas aleatoriamente para gerar um intervalo dentro de duas partículas chamadas de pais (p_1 e p_2), em seguida duas novas partículas são criadas, denominadas filhos (c_1 e c_2), trocando todos os *jobs* que estavam dentro do intervalo de uma partícula pai pelos *jobs* do outro pai. Esta técnica está elucidada na Figura 6.

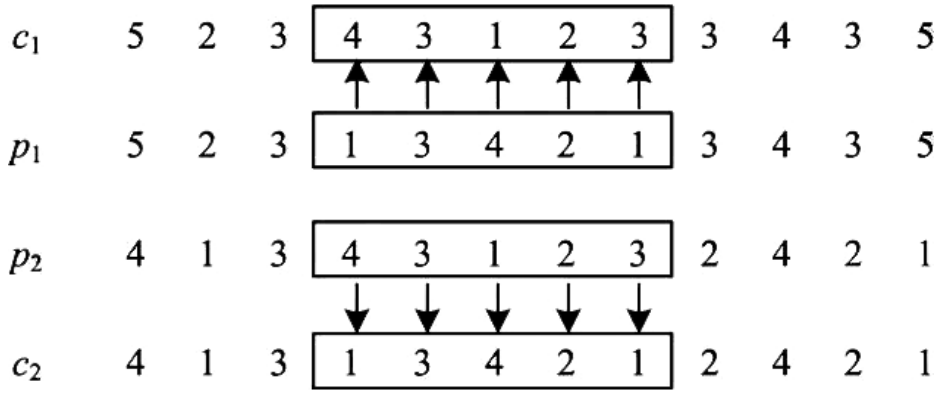

Figura 6: Crossover para geração de novas partículas
Fonte: Adaptado de (ZHANG et al., 2009).

Já na permutação da codificação *B-string*, também são selecionados dois pais do enxame, denotados por p_1 e p_2, e duas cópias chamadas de filhos c_1 e c_2. Após isso todos os *jobs* são divididos em dois grupos JG_1 e JG_2 aleatoriamente. Qualquer elemento em p_1/p_2 que pertença à JG_1/JG_2 fica mantido nos filhos nas mesmas posições, e os que não pertencem são deletados. As posições vazias de c_1/c_2 são preenchidas por elementos de p_1/p_2 que estão contidos em JG_2/JG_1. Para melhor entendimento observe a Figura 7.

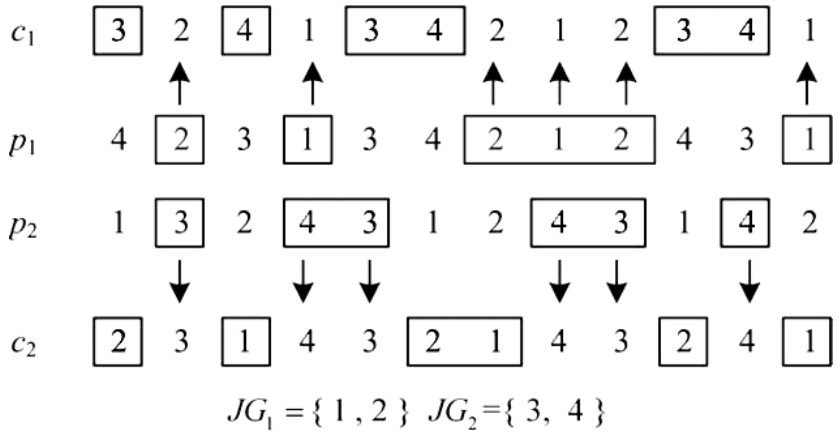

Figura 7: Permutação das operações de uma partícula
Fonte: Adaptado de (ZHANG et al., 2009).

Foram utilizadas quatro bases de testes, as mesmas utilizadas na explanação da pesquisa de KACEM; HAMMADI; BORNE (2002). Em comparação ao trabalho de XIA; WU (2005), a melhora está presente no tempo de processamento de todos os recursos.

3.6 Aproximação Pareto em PSO Multiobjetivo (MOPSO)

Um estudo em forma de Pareto utilizando algoritmo híbrido PSO e Busca Local para resposta do FJSP foi proposto por MOSLEHI; MAHNAM (2011), porém com a concepção multiobjetivo inerente ao algoritmo PSO (*Multi-Objective Particle Swarm Optimization* - MOPSO), o que difere do modelo multiobjetivo utilizando soma dos pesos da minimização de cada função-objetivo.

Transformar PSO em MOPSO requer uma redefinição da regra de seleção do melhor resultado global (*gB*), a fim de obter uma solução ótima que irá guiar o algoritmo. Naturalmente, essa seleção é feita a partir do conjunto de soluções ótimas Pareto. Este problema é difícil e importante para obtenção de convergência e diversidade de soluções. Uma vez que cada partícula do enxame deve selecionar uma das soluções ótimas do Pareto como sua influência global própria, não mais uma única solução *gB* influenciará todas as partículas, sendo a solução selecionada chamada de melhor guia local desta partícula. Assim, a parte mais importante do MOPSO é a determinação do melhor guia local das partículas. Neste método, a estratégia de

elitismo é considerada a fim de não perder as soluções não-dominadas durante o decorrer das gerações. O PSO multiobjetivo utiliza para encontrar as soluções guias das partículas, o método sigma introduzido por MOSTAGHIM; TEICH (2003).

A Busca Local (LS) segue o intuito dos trabalhos anteriores de explorar a combinação de operações a partir de um espaço de busca mais limitado (mas promissora), utilizando aleatoriamente soluções geradas baseadas em uma heurística. Este algoritmo começa com a identificação do caminho crítico na solução obtida no subproblema de roteamento. Então, operações críticas são atribuídas a recursos produtivos com menor tempo de processamento ou menor tempo de completude, considerando as limitações de tempo de liberação, de precedência e de capacidade. Este procedimento é refeito até que o *makespan* melhore.

Considerando a possibilidade de estagnação em ótimos locais, foi projetado um operador de mutação, baseado em dois parâmetros m_1 e m_2, implantado em dois estágios. Este operador escolhe as partículas com uma probabilidade de m_1, e aleatoriamente muda cada dimensão da partícula com uma probabilidade de m_2. Ainda com o objetivo de evitar a estagnação visando melhorias na vizinhança, foi desenvolvido um método que, se passadas g_1 iterações e as soluções ótimas Pareto não melhorarem, então a vizinhança global irá substituir a local, e também se passadas g_2 iterações e a solução não adquirir melhora, uma nova população de partículas será geradas aleatoriamente.

Todo o procedimento pode ser mais bem entendido com a Figura 8 que mostra a interação entre o algoritmo PSO e LS, mostrado abaixo.

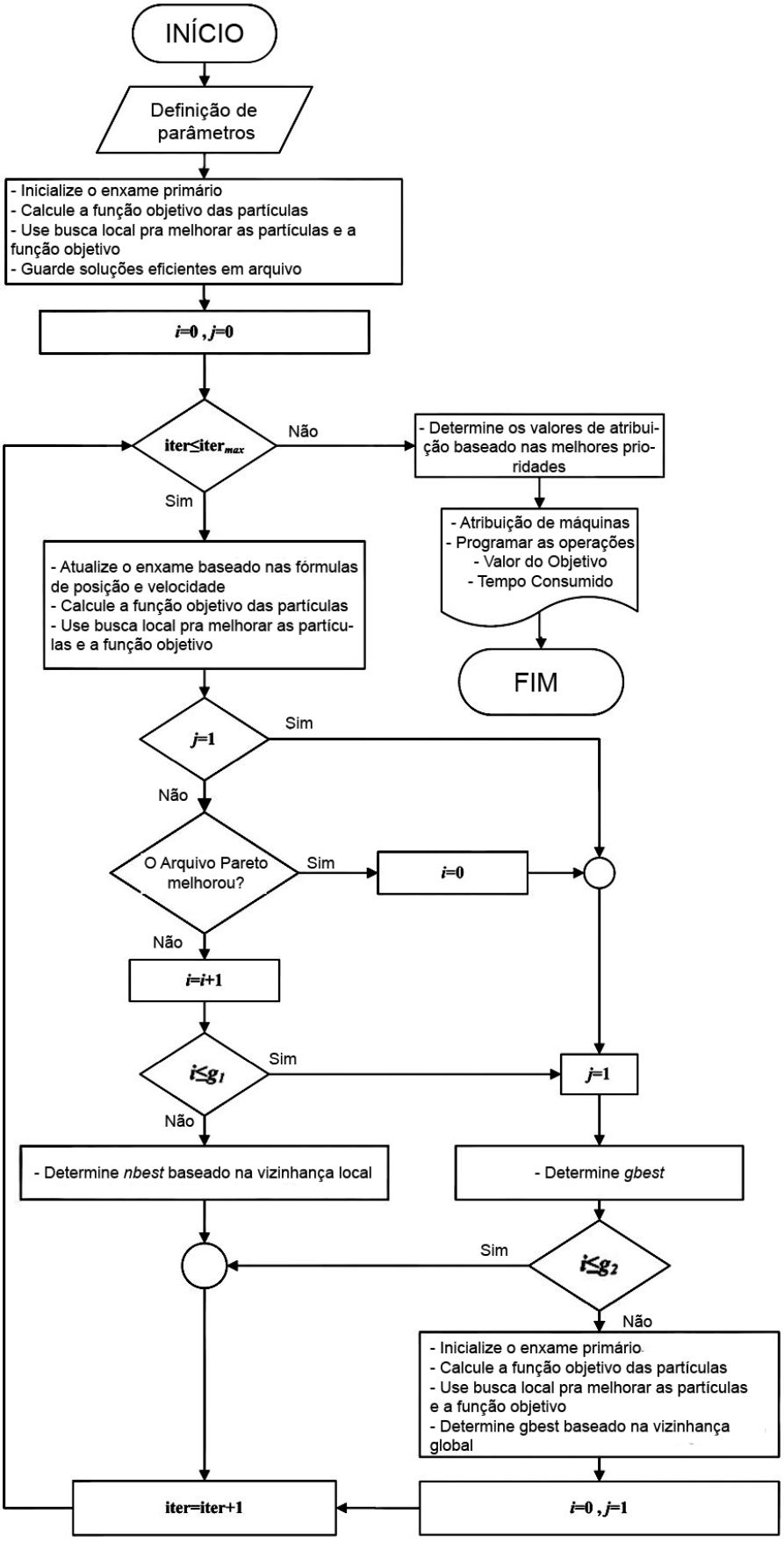

Figura 8: Fluxograma do algoritmo híbrido MOPSO
Fonte: Adaptado de MOSLEHI; MAHNAM (2011).

3.7 PSO Discreto (DPSO) e SA

Este trabalho desenvolvido por SHAO et al. (2013) tem como base a pesquisa híbrida de PSO e SA feita por XIA; WU (2005), porém utilizando apenas espaço de busca discreto, motivado por três principais aspectos negativos do algoritmo PSO multiobjetivo introduzidos por COELLO COELLO; REYES-SIERRA (2006), as propostas perante os aspectos são:

1. Como as partículas são designadas líderes (melhor local, melhor global) de modo a dar preferência às soluções com tempos menores em relação às soluções com tempos maiores?

   Com a intenção de avaliar precisamente uma partícula, o conceito de *Rankeamento* baseado em Pareto é introduzido. Utilizando uma função de aptidão binária (discreta), resultando assim na maior probabilidade de seleção de uma partícula com *fitness* inferior como líder.
2. Como as melhores soluções encontradas são preservadas?

   Uma nova estratégia a qual retém todas as partículas dos melhores locais, sem usar um conjunto novo para guardar estes indivíduos.
3. Como a diversidade é mantida no enxame para evitar a convergência em ótimos locais?

   Com uma abordagem de distância na aglomeração das partículas, unindo a atualização da velocidade por dependência de similaridade e a possibilidade do SA aceitar soluções piores.

Tanto o PSO quanto o SA utilizam o conceito de *Rankeamento* baseado em Pareto, ou seja, a partir de um conjunto Pareto de soluções (soluções boas, mas que possuem algum conceito negativo), são construídos *ranks* dessas soluções para influenciar as novas soluções encontradas no algoritmo. Este trabalho apresentou uma melhora no terceiro objetivo do cenário 15x10, decaindo de 93, resultado encontrado na pesquisa de (ZHANG et al., 2009), para 91. Esta e as demais comparações podem ser vistas no item 3.9.

3.8 PSO Multiagente

Para finalizar os estudos sobre a resolução do FJSP utilizando o algoritmo PSO foi analisado o trabalho de NOUIRI et al. (2015), onde foi desenvolvido a meta-heurística de Enxame de Partículas com o auxílio do conceito multiagente, que pode ser explanado segundo YAN CHEN; ZENG-ZHI LI; ZHI-WEN WANG (2004), como um sistema artificial composto de uma população de agentes autônomos, que cooperam entre si para alcançar objetivos em comum, enquanto simultaneamente cada agente aspira objetivos individuais.

Um exemplo retirado da própria pesquisa mostra que dois tipos de agentes interagem entre si, sendo eles o *Execute agent* (EA), que cria uma subpopulação e processa todo o algoritmo mandando seu melhor global para o *Syncronization agent* (SA), o qual está responsável por inserir este novo melhor global encontrado no processamento do seu algoritmo PSO, como visto na Figura 10, cujo procedimento de subpopulação foi dividido em três, EA1, EA2 e EA3.

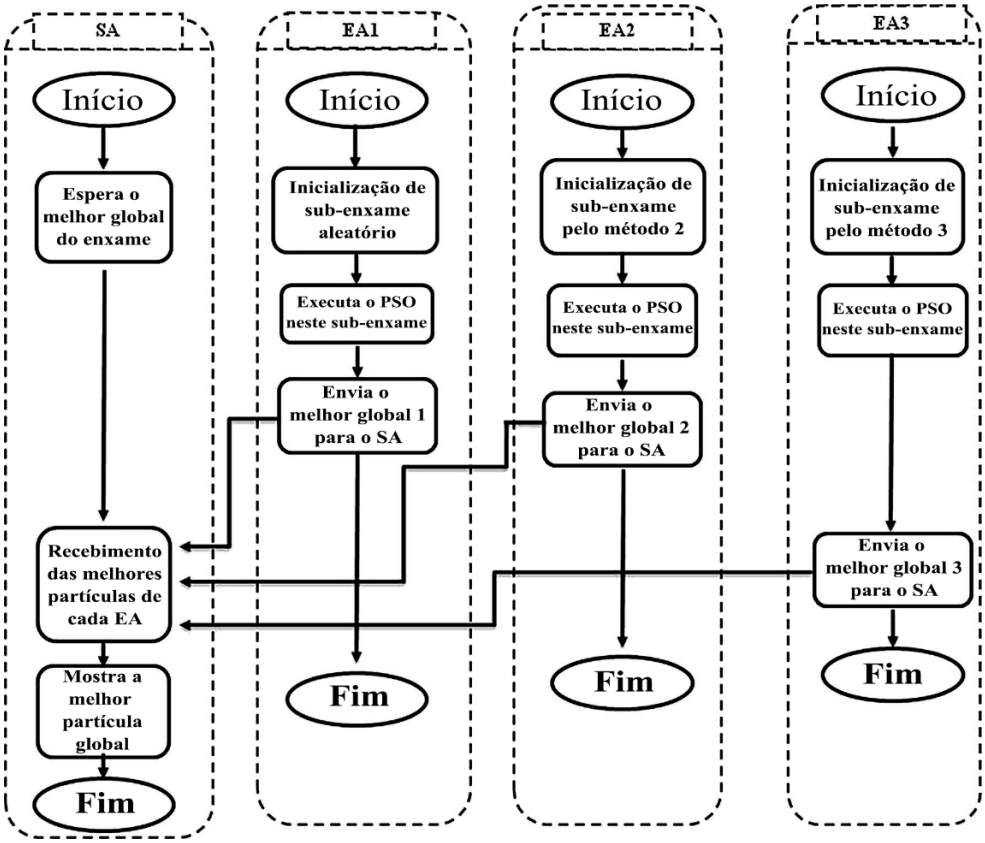

Figura 9: Fluxograma do algoritmo PSO Multiagente
Fonte: Adaptado de (NOUIRI et al., 2015).

Os resultados encontrados com este PSO unido com o conceito multiagente, oscilaram ora como ótimos (encontrados desde a pesquisa de ZHANG et al. (2009)), ora um pouco acima do ótimo. Mostrando assim que até a presente data novas técnicas de aperfeiçoamento do PSO são plausíveis para a resolução do FJSP, uma vez que esta meta-heurística concebe bons resultados.

3.9 Pontos Importantes e Resultados

Deve-se frisar primeiramente que ao decorrer dos anos o poder computacional evoluiu, resultando assim na possibilidade de inserção de novas complexidades para o FJSP, como os objetivos dos trabalhos analisados que passaram do conceito mono-objetivo para o multiobjetivo, e os cenários que puderam receber mais operações, essas modificações e complexidades visam torná-lo cada vez mais próximo dos problemas reais. Porém mesmo que o poder computacional seja suficiente para as pesquisas abordadas, alguns aspectos importantes devem ser questionados a fim de obter uma busca mais aprofundada no espaço de soluções e melhora no tempo de resposta dos algoritmos.

No trabalho de KACEM; HAMMADI; BORNE (2002) e de XIA; WU (2005), foram atribuídos valores altos para os tempos de processamentos nos modelos de resolução do FJSP em caráter parcialmente flexível, com o objetivo de inferiorizar a relevância dos roteiros que possuem recursos produtivos que não processam determinadas operações. Este procedimento simplifica o tratamento de novos roteiros, uma vez que não é necessária a verificação se existe recursos produtivos que não processam as operações dadas, porém são gastas muitas iterações calculando a aptidão de soluções irrelevantes, iterações essas que poderiam ser destinadas a uma convergência mais rápida para a solução ótima.

Em um âmbito geral, algoritmos PSO possuem uma provável estagnação decorrente de seu comportamento sociocognitivo, mesmo tendo este problema sido atenuado por MOSLEHI; MAHNAM (2011), ainda está fortemente inserido em sua pesquisa, pelo fato de que uma nova população gerada aleatoriamente pelo procedimento supracitado pode convergir para o mesmo mínimo local ou outro.

Desde a proposta de KACEM; HAMMADI; BORNE (2002), são utilizados a mesma base de codificação dos *jobs* para fim de comparação dos algoritmos com soma ponderada dos objetivos. Na Tabela 3 consta uma comparação que contém os algoritmos citados nesta seção, é possível notar que a partir da pesquisa sobre o MOPSO, houve uma estagnação de ótimos, porém ainda há uma variação na quantidade de partículas e iterações usadas. O tempo de processamento não foi comparado. Os problemas contidos na base supracitada estão divididos em níveis de complexidade e por tipo de cenário, como por exemplo o problema 8x8, que representa um cenário P-FJSP e de média escala, pois contém 8 *jobs*, que devem ser processados em 8 máquinas. Já os problemas 10x10 e 15x10 são considerados de larga escala, sendo ambos cenários T-FJSP. A base de dados será explanada detalhadamente no capítulo 5.

Somente a pesquisa de NOUIRI et al. (2015), dentre as usadas como referência, foi desenvolvida utilizando a abordagem integrada, ou seja, sem a separação do problema em dois subproblemas, o de roteamento e o de programação, resolvendo-os simultaneamente. Enquanto as outras pesquisas referenciadas neste trabalho utilizam a abordagem hierárquica, destinando um algoritmo de busca global para o subproblema de roteamento e um algoritmo de busca local para o subproblema de programação.

Tabela 3: Comparativo dos trabalhos analisados que utilizaram a base de dados de KACEM; HAMMADI; BORNE (2002).

nxm	Objetivo	AL + CGA	PSO + SA	PSO + TS	MOPSO	hDPSO	Dist PSO
8x8	C_m	15	15	14	14	14	17
	W_m	-	12	12	12	12	-
	W_t	75	75	77	77	72	-
10x10	C_m	7	7	7	7	7	8
	W_m	5	6	6	5	5	-
	W_t	41	44	43	43	43	-
15x10	C_m	23	12	11	11	11	-
	W_m	11	11	11	11	11	-
	W_t	91	91	93	91	91	-
Max: Iterações - População	-	500 - 100	100 - 50	100 - 50	100-100	100* - 500	200 - 100

*valor aproximado

Capítulo 4

PROPOSTA DO TRABALHO

4.1 Formulação do Problema

Como aludido no Capítulo 2, o Planejamento e Controle da Produção (PCP), resulta em problemas que carecem de mecanismos para sua resolução, sendo um desses problemas, o FJSP, elegido para o âmbito desta pesquisa, é um dos modelos mais complexos e realistas de prática fabril. O fato de que O FJSP possui um conjunto de *jobs* que deve ser processado em um conjunto de recursos produtivos, almejando a minimização dos objetivos, resulta em um problema NP-difícil, cuja resolução não é possível deterministicamente em tempo computacional polinomial (GAREY; JOHNSON, 1979). Neste contexto, o modelo de otimização sugerido, PSO híbrido, é indicado para minimização dos objetivos propostos.

O modelo de resolução do FJSP varia conforme os objetivos aderidos, restrições, e cenários, sendo necessário o desenvolvimento de um algoritmo capaz de prover uma solução em tempo hábil. Diante de tais indicadores, o algoritmo proposto foi projetado para lidar de forma dinâmica com as diferentes priorizações de cada objetivo, uma vez que um objetivo primado por um PCP pode não ser o mesmo de outro PCP, como também possuir adaptação aos diferentes tipos de cenários, P-FJSP e T-FJSP, respeitando as restrições encontradas na literatura. O algoritmo e mecanismos desenvolvidos para resolução do FJSP são explanados nas seções seguintes.

Este trabalho apresenta uma comparação entre três meta-heurísticas auxiliares, TS, SA e RRHC somado a melhorias no PSO para a minimização do problema *job-shop* flexível multiobjetivo, cujo procedimento pode ser explanado da seguinte maneira: Há um conjunto de n *jobs* e um conjunto de m recursos produtivos. M denota o conjunto de todos os recursos. Cada *job* i consiste de uma sequência de n_i operações. Cada operação $O_{i,j}(i = 1, 2, \dots, n; j = 1, 2, \dots, ni)$ de *job* i tem que ser processada em um recurso M_k pertencente a um conjunto de recursos produtivos compatíveis $M_{i,j}$ (para $M_k \in M_{i,j}, M_{i,j} \subseteq M$). A execução de cada operação requer um recurso selecionado a partir de um conjunto de recursos disponíveis. Para completar, o FJSP precisa estipular o tempo de início e fim para cada operação, as atribuições e a sequência das operações que satisfazem as restrições impostas. Se há $M_{i,j} \subset M$ para pelo menos uma operação, isto é chamado de flexibilidade parcial FJSP (P-FJSP), subentende-se que nem todos os recursos do conjunto de recursos estão aptos a processar os *jobs*, enquanto se $M_{ij} = M$ para cada operação, isto é chamado de flexibilidade total, cenário onde todos os recursos estão aptos a processar todos os *jobs* (KACEM; HAMMADI; BORNE, 2002).

Para a validação da solução do problema foram estipuladas três funções-objetivo a serem minimizadas, observadas também nas pesquisas do capítulo 3, são elas:

f_1 = *Makespan*, ou seja, o tempo total do início do processamento até o processamento da última operação;

f_2 = Carga de trabalho dos recursos produtivos de forma isolada a fim de balancear o tempo de trabalho;

f_3 = Carga de trabalho de todas os recursos, ou seja, a soma do tempo gasto em todas as operações, mesmo do que foi feito em paralelo.

Neste projeto foi adotado a estratégia de soma ponderada dos valores obtidos nas três funções-objetivo descritas anteriormente. Os pesos dessa soma variam de acordo com o cenário utilizado e suas prioridades. Ainda para a compreensão da proposta, foram definidas as restrições com base nos trabalhos analisados no capítulo 3, estas são:

1) Cada operação não pode ser interrompida durante seu processamento em um recurso;
2) Cada recurso pode executar no máximo uma operação por vez (restrição de recursos);
3) As restrições de precedência das operações de um *job* podem ser definidas para qualquer par de operações;
4) O tempo de configuração dos recursos produtivos e transportes entre eles não serão considerados;
5) Os recursos são independentes uns dos outros;
6) Os *jobs* são independentes uns dos outros.

4.2 Materiais e Métodos

A metodologia empregada no desenvolvimento deste projeto inicia-se com um processo de revisão sistemática identificando trabalhos relevantes existentes na literatura. A abordagem de modelagem escolhida para resolução do FJSP, foi a hierárquica, cujo procedimento divide o problema em dois, o subproblema de roteamento e o subproblema de programação.

No subproblema de programação, foi constatado o uso de uma meta-heurística complementar para o PSO, uma vez que algoritmos baseados em busca local são mais indicados para este tipo de problema, por se tratar unicamente de permutação das operações dos roteiros, resultando assim em um espaço de busca menor e mais simples. Os algoritmos complementares escolhidos para esta pesquisa foram o SA, TS e RRHC, as escolhas do SA e TS foram corroboradas pela fomentada presença e eficácia comprovada em estudos sobre FJSP, enquanto o RRHC foi escolhido com o intuito de ser uma alternativa eficiente, simples e rápida, mesmo com presença praticamente nula em pesquisas realizadas até o momento sobre FJSP.

Todas as meta-heurísticas auxiliares são aplicadas de maneira bem próxima a sua forma padrão, apenas com a adaptação da solução, para que trabalhem com o indivíduo (partícula gerada no PSO) em representação vetorial.

Para o desenvolvimento e aplicação do algoritmo híbrido foi utilizado a ferramenta MATLAB 2015b, por ser uma plataforma de rápido desenvolvimento,

robusta, com forte representação matricial, o que simplifica o cálculo de vetores únicos e polidimensionais (HAHN; VALENTINE, 2013).

4.3 Aplicação da abordagem

Neste capítulo são apresentados alguns mecanismos que auxiliam a aplicação da abordagem, visando aperfeiçoar os resultados obtidos, seja na diminuição de tempo de processamento ou aquisição de resultados melhores. Para a implementação da abordagem hierárquica, foi utilizado o conceito de hibridização, em que o PSO mapeia uma solução de roteamento em forma de partícula e transfere para o algoritmo auxiliar de busca local para que haja a programação do mesmo. Esta partícula representa um vetor de recursos que é transformado em um vetor de níveis para aumentar a abrangência da solução.

A implementação do PSO visa o controle de estagnação utilizando o mecanismo de não repetição de uma partícula, uma vez que seu aprendizado sociocognitivo advindo da combinação de pesquisa local (por experiência própria) e pesquisa global (por experiência dos vizinhos), se direcionado para uma única solução, fornece um aumento significativo na probabilidade de estagnação.

Além disto, destaca-se o tratamento dos cenários P-FJSP, posto que a utilização de tempos com valores enormes para os recursos que não processam determinadas operações, como na pesquisa de KACEM; HAMMADI; BORNE (2002), que foi utilizado 999, ocasiona no processamento de partículas que não irão influenciar o algoritmo em grau nenhum, pois essas serão descartadas por terem um tempo exorbitante.

Outras técnicas que também contribuem com a precisão do algoritmo híbrido proposto foram delineadas, como a codificação estocástica e a seleção de vizinhança do subproblema de programação.

4.3.1 Representação dos subproblemas

Uma boa representação da codificação das entradas dos algoritmos tem grande influência em todo o processo de otimização, pois segundo ZHANG et al.

(2009), se os mecanismos de mapeamento do problema e geração de solução forem bem desenvolvidos, é possível obter bons resultados em tempos aceitáveis. Em primeira instância, fundamentado no trabalho de XIA; WU (2005), foi escolhido para a representação do indivíduo no subproblema de roteamento o mecanismo de vetor baseado em níveis de operação (ver Figura 10), em razão de que cada coluna ilustra as operações de forma sequencial, e o valor contido na célula representa o nível do recurso escolhido, sendo o menor nível correspondente ao menor tempo de processamento. Já a partícula baseada em operações, tem contida em cada célula o recurso para processamento da operação.

A Figura 10, demonstra que o valor 3 contido na coluna da operação $O_{1,3}$ no vetor de partículas baseado em operação, na verdade representa o recurso M_3, enquanto o vetor de partículas baseado em níveis de operação tem o valor 1, que representa o menor tempo de processamento que $O_{1,3}$ pode ter, 7, este tempo é disposto pelo recurso M_2 e M_3, porém o M_3 foi selecionado para a partícula ilustrada. Esta partícula é a mesma partícula usada na Figura 1 para exemplificação do problema 3x3 contido na Tabela 1, todavia está elucidado da forma com que o algoritmo processa e não mais por arcos conjuntivos. Este vetor de níveis será convertido em dois ou mais vetores de mesma magnitude, com as possibilidades de combinação dos recursos que possuem o mesmo nível para serem processados paralelamente, dando assim mais abrangência com uma única partícula.

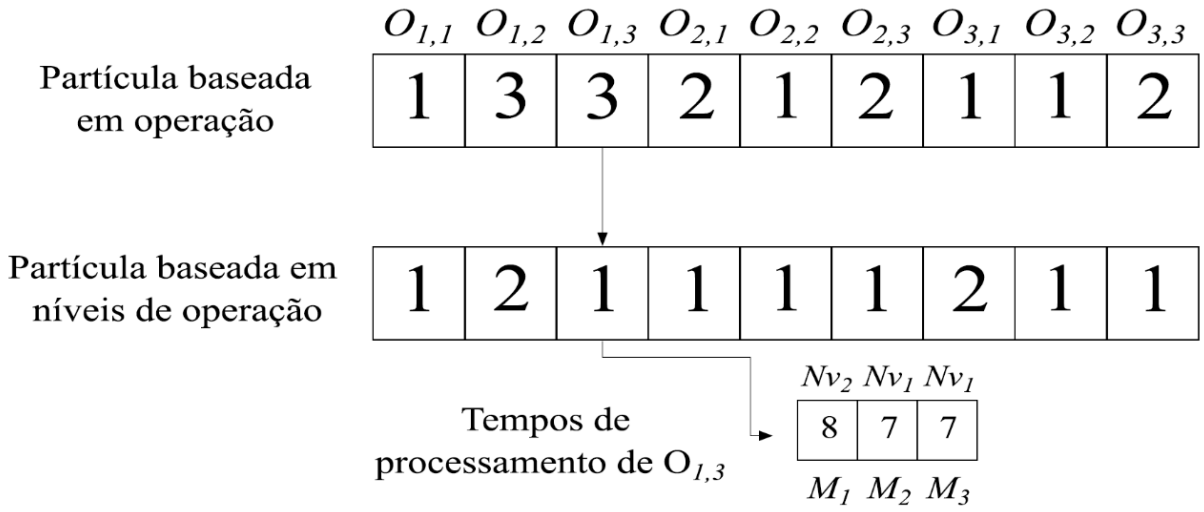

Figura 10: Representação vetorial da partícula baseada em níveis de operação.

Na codificação da forma supradita, quando ocorre o caso de um mesmo nível possuir dois recursos com tempos iguais, vide Figura 10, onde M_2 e M_3 possuem o mesmo tempo, neste caso, sendo o nível 1 escolhido para a operação $O_{1,3}$, duas partículas seriam criadas e processadas paralelamente para que cada recurso fosse usado nesta operação. É nítido que se outras operações contiverem níveis com múltiplos recursos, o número de combinações deverá aumentar, e para controlar uma possível explosão de possibilidades, foi definido um limite de 30 variações das partículas por iteração.

Após a determinação das partículas baseadas em operação com os valores dos recursos em cada célula, estas são enviadas para o subproblema de programação, entretanto, serão usadas de outra forma, elas devem ser transformadas em codificações baseadas em recursos, na qual cada coluna representa um recurso produtivo, tendo como conteúdo um vetor com todas as operações que são processadas neste recurso, a ordem desse vetor é a chave do subproblema de programação, consequentemente o vetor baseado em recurso facilita a ordenação pelos algoritmos de busca local (XIA; WU, 2005). O modelo dessa transformação é visto na Figura 11, em que o recurso M_3 processa as operações $O_{1,2}$ e $O_{1,3}$, como o algoritmo classifica as operações por números de um até a quantidade total de operações, no recurso M_3 estão contidos os valores 2 e 3. Conforme visto na Tabela 1, este problema possui três operações por *job*, logo, o valor máximo assumido foi 9.

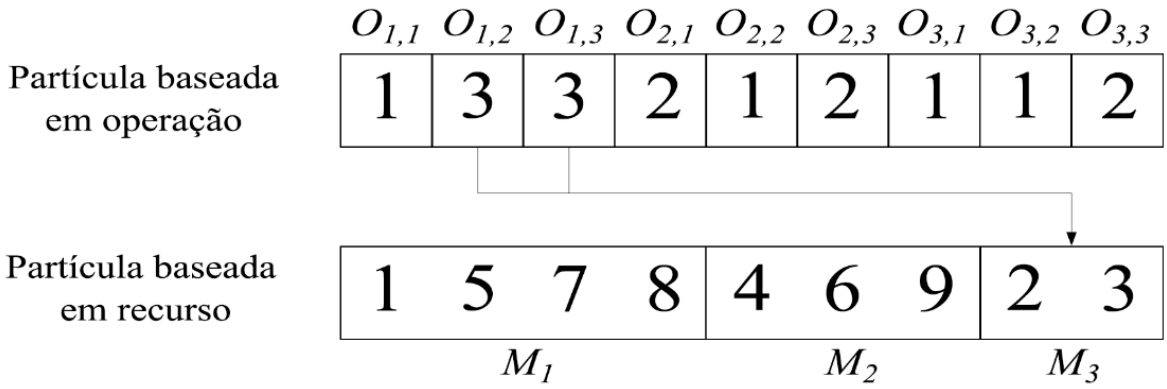

Figura 11: Transformação da partícula baseada em operação para baseada em recurso.

4.3.2 Controle antiestagnação

Conforme o aprendizado do PSO, sua exploração do espaço de solução dá-se pela influência do melhor valor local (*pB*) e global (*gB*), porém se ambos detiverem o mesmo valor, a dispersão das novas partículas será fortemente induzida pelo melhor valor corrente, acarretando assim em uma possível estagnação de mínimo local, o que fará o algoritmo gerar, na maioria das vezes, a mesma resposta.

Uma solução proposta por este trabalho foi a de criar um registro que armazena todas as soluções geradas pelo algoritmo para que a cada nova iteração, se for gerada uma solução contida nesse registro, um *job* é escolhido aleatoriamente para a troca do nível em seu roteiro, esse procedimento é feito enquanto a solução ainda pertencer ao registro. Caso uma solução contenha mais do que 30 variações de partículas, ela só entrará no registro de soluções usadas, desde que todas as variações sejam processadas pelo algoritmo.

4.3.3 Codificações estocásticas

Na definição da população inicial é utilizado um método que gera todas as partículas de forma aleatória, porém com um viés na seleção, que pode ser análogo a uma roleta, onde os níveis menores gozam de maiores espaços na roleta de seleção, dando assim mais probabilidade de partículas com tempos menores serem geradas sem descartar os *outliers* (aquilo que está fora de padrão). O mesmo ocorre para a substituição de um nível em uma partícula que já foi utilizada pelo PSO (descrito no item anterior), um novo nível é concebido a partir da mesma seleção probabilística da codificação inicial.

Sabendo-se que há uma ordem de precedência das operações a ser respeitada, as operações de número $j = n_i$ e $j = n_i - 1$, ou seja, a penúltima e última operação de cada *job,* são comumente colocadas no fim da programação da produção de uma solução ótima, este fator foi observado nas pesquisas sobre FJSP, por conseguinte, foi sugerido por este trabalho uma prévia alocação das operações nos recursos, de forma que as últimas operações de um *job* sejam dispostas no fim da programação de um recurso, e as primeiras no começo, para que posteriormente seja submetido a resolução do subproblema de programação. A Figura 12 exemplifica a mudança de posição das operações do recurso M_1, foram alocados da ordem de

processamento de $5 \rightarrow 7$ para $7 \rightarrow 5$, pois o número 7 representa a primeira operação do *job* 3, já o número 5 a segunda operação do *job* 2.

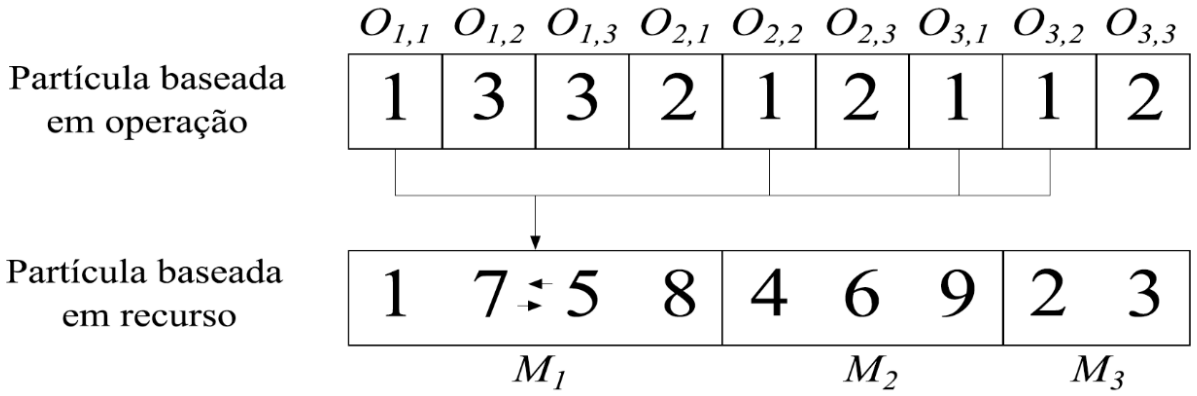

Figura 12: Alteração do posicionamento das operações na partícula baseada em recurso.

4.3.4 Tratamento de P-FJSP

Sugerido por KACEM; HAMMADI; BORNE (2002), os recursos produtivos que não pertencem ao conjunto de recursos que processam uma determinada operação, devem ter seu tempo de processamento elevado a próximo do 'infinito', todavia mesmo esta sendo uma interpretação bem simples de ser implementada, resulta na utilização de tempo de processamento do algoritmo na parte de programação do roteamento de forma desnecessária, uma vez que este roteiro deveria ser desconsiderado. Um exemplo do porque isto ocorre está na atualização das posições das partículas no PSO, quando o deslocamento faz com que um nível que não pertence ao conjunto de níveis que processam uma determinada operação seja selecionado, e isto ocorre eventualmente uma vez que quesitos aleatórios estão envolvidos.

Para solucionar este problema, é endereçado para a operação o maior nível possível, visto que se houve um endereçamento de um nível inexistente, este excedeu o limite dos níveis, logo, o maior nível é cabível. Este procedimento garante que todas as soluções geradas pelo algoritmo sejam factíveis.

4.3.5 Função de vizinhança das meta-heurísticas auxiliares

A função de vizinhança de cada meta-heurística é amparada pela utilização de caminhos críticos advindos da função de aptidão do algoritmo híbrido. Um caminho crítico é a sequência de atividades de um projeto que somadas têm duração total do projeto (STELTH; ROY, 2009). Deste modo, um novo vizinho é gerado da realocação das operações que estão contidas em um caminho crítico. Seja G o conjunto de todas as operações contidas em caminhos críticos de uma solução, P é um caminho crítico escolhido aleatoriamente. $P \in G$. Para cada meta-heurística é elaborado um vetor contendo as operações de um caminho crítico aleatório P, como visto na Figura 13. É escolhido então uma operação neste vetor P de forma aleatória para ter sua posição substituída com outra operação no mesmo recurso, esta função diminui o tamanho da vizinhança e aprimora os resultados (MASTROLILLI; GAMBARDELLA, 2000).

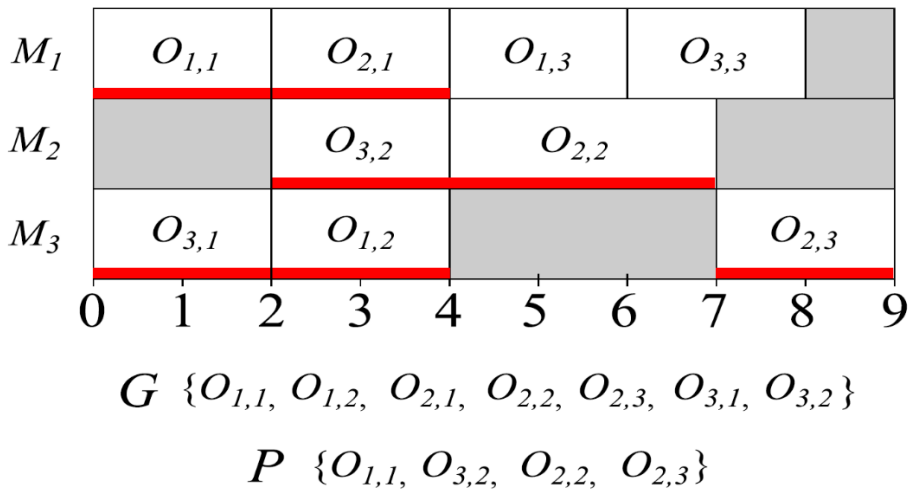

Figura 13: Exemplificação de caminhos críticos representados em um Gráfico de Gantt

A troca de posições é feita da seguinte maneira: defina $op1$ como a operação escolhida aleatoriamente em P, por exemplo $O_{1,1}$, como mostrado na Figura 13, e k o vetor que representa o recurso contendo as operações que são processadas nele, inclusive $op1$, neste caso M_1. Origina-se então $k = \{O_{1,1}, O_{2,1}, O_{1,3}, O_{3,3}\}$, se a posição p de $op1$ é igual o tamanho de k, troque $op1$ com a operação em $p - 1$, senão troque

$op1$ com a operação em $p + 1$. Como p de $op1$ é 1 e o tamanho de k é 4, a substituição é feita com a operação contida em $p + 1$, resultando em $k = \{O_{2,1}, O_{1,1}, O_{1,3}, O_{3,3}\}$.

4.4 PSO Híbrido Proposto

A combinação da experiência própria de uma partícula e de sua vizinhança (pB), somada à experiência de todas as partículas (gB), caracterizam o PSO como algoritmo de busca global, ao passo que os algoritmos SA, TS e RRHC preenchem a lacuna de um modelo simples para a resolução do subproblema de programação em tempo favorável. O fluxograma do algoritmo híbrido pode ser visto na Figura 14.

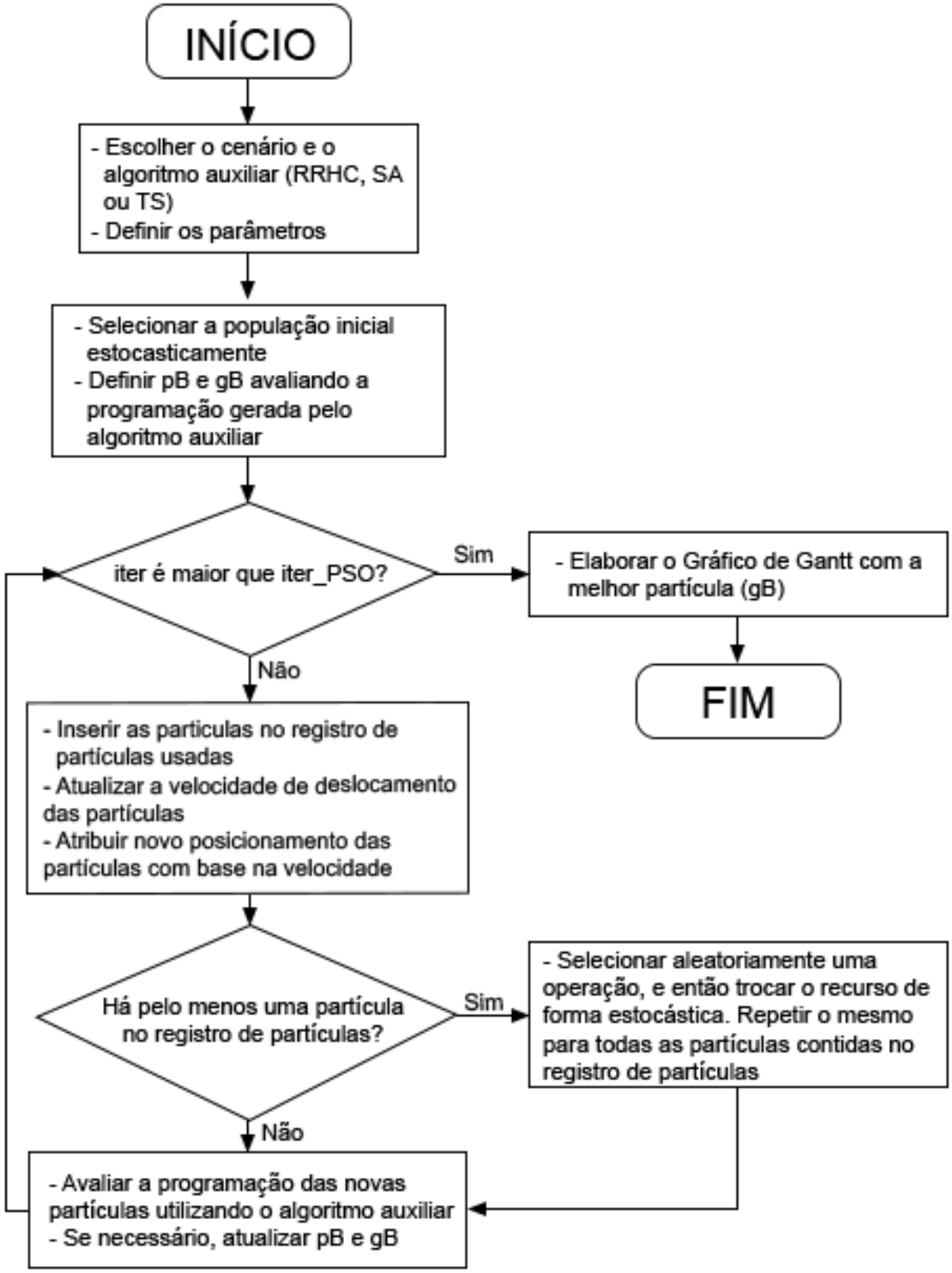

Figura 14: Fluxograma do Algoritmo Híbrido Proposto.

Capítulo 5

ESTUDO DE CASO E ANÁLISE DE EXPERIMENTOS

5.1 Implementação

Neste capítulo serão apresentadas os cenários que foram otimizados com o algoritmo híbrido proposto PSO, utilizando três meta-heurísticas auxiliares, SA, TS e RRHC. A base de dados foi retirada da pesquisa de KACEM; HAMMADI; BORNE (2002), embora tenha sido proposta há mais de 10 anos, ainda é utilizada até o presente momento, como pode ser visto na pesquisa de (NOUIRI et al., 2015), (HUANG; TIAN; JI, 2016). Cada algoritmo híbrido foi testado 10 vezes em cada problema, ou seja, 30 testes por problema. Todos os testes são apresentados e comentados sequencialmente, há uma comparação da performance do algoritmo híbrido proposto com cada meta-heurística auxiliar, e com pesquisas similares sobre FJSP. Também foi realizado um informativo dos resultados das hibridizações com o mínimo encontrado, média e desvio padrão, mostrado na Tabela 10. O algoritmo híbrido proposto foi desenvolvido na plataforma MATLAB 2015b em um computador pessoal com 3.3 GHz de CPU, 6 núcleos e 8GB de RAM.

Quatro instâncias representativas baseadas em dados práticos foram selecionadas, os parâmetros das bases para entendimento são: Número de *jobs* (n), quantidade de recursos produtivos (m), e a operação $O_{i,j}$ (sendo i o *job* atual e j a operação deste *job*). Todos os parâmetros utilizados no algoritmo são baseados nas pesquisas do Capítulo 3, somado à experiências empíricas para os parâmetros de

iteração e população, isto é, foram ajustados perante a conjuntura e necessidade do estudo.

A configuração dos parâmetros do algoritmo híbrido foi dada como segue: c_1 e c_2 = 2 (PSO, (ZHANG et al., 2009)) ; $weight$ = 0.4 ~ 1.2 (PSO, (XIA; WU, 2005)); α = 0.9 ~ 0.95 (SA, (XIA; WU, 2005)); iterações por temperatura = 10 (SA); $listLength$ = 5 (TS); iterações = 300 (TS); $n\_restart$ = 50 (RRHC), iterações = 500 (RRHC). A função de aptidão do algoritmo é uma soma ponderada dos três objetivos, C_k, W_m, e W_t, que respectivamente possuem os pesos: 0.8, 0.11 e 0.09, estes foram adotados de forma empírica, dado que em nenhum dos estudos usados como base foram apresentados os seus pesos.

O primeiro objetivo Ck, comumente recebe o maior peso por representar o tempo real da produção, ou seja, o tempo decorrido entre o começo da primeira operação e o término da última, tornando o algoritmo mais compatível às expectativas de uma empresa. No entanto, a pequena discrepância, que resulta na prioridade do objetivo Wm em relação ao Wt, é justificada pela necessidade em evitar a maior influência de Wt em comparação ao Wm, posto que seu valor é maior por representar a soma de todos os tempos de processamento. E por conseguinte, para esta base de dados selecionada (KACEM; HAMMADI; BORNE, 2002), priorizar uma solução que minimizou o tempo no objetivo Wm, favorece a busca por melhores soluções.

Os parâmetros que controlam a quantidade de iterações e número de partículas necessárias do PSO para a resolução do FJSP, são definidos para cada cenário.

5.1.1 Cenário 4x5

Este teste apenas foi executado para comprovar que o algoritmo também lida com cenários de pequena escala, neste caso, como na primeira iteração com todas as meta-heurísticas o ótimo já havia sido atingido, não foi feito comparação da convergência das soluções. O cenário é do tipo totalmente flexível, conta com 12 operações no total, 4 *jobs* que utilizam 5 recursos produtivos. Os tempos de processamento podem ser encontrados na Tabela 4.

Tabela 4: Cenário 4x5 extraído de KACEM; HAMMADI; BORNE (2002).

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5
1	$O_{1,1}$	2	5	4	1	2
	$O_{1,2}$	5	4	5	7	5
	$O_{1,3}$	4	5	5	4	5
2	$O_{2,1}$	2	5	4	7	8
	$O_{2,2}$	5	6	9	8	5
	$O_{2,3}$	4	5	4	54	5
3	$O_{3,1}$	9	8	6	7	9
	$O_{3,2}$	6	1	2	5	4
	$O_{3,3}$	2	5	4	2	4
	$O_{3,4}$	4	5	2	1	5
4	$O_{4,1}$	1	5	2	4	12
	$O_{4,2}$	5	1	2	1	2

Para a otimização deste cenário foi utilizado: Número de iterações PSO = 5; Números de partículas = 5. A solução apresentada pelas três meta-heurísticas possui o mesmo valor do ótimo encontrado na literatura, $C_k = 11$, $W_m = 10$, $W_t = 32$, como mostra o Gráfico de Gantt da Figura 15. O tempo de execução foi praticamente o mesmo para as três meta-heurísticas, cerca de 8 segundos.

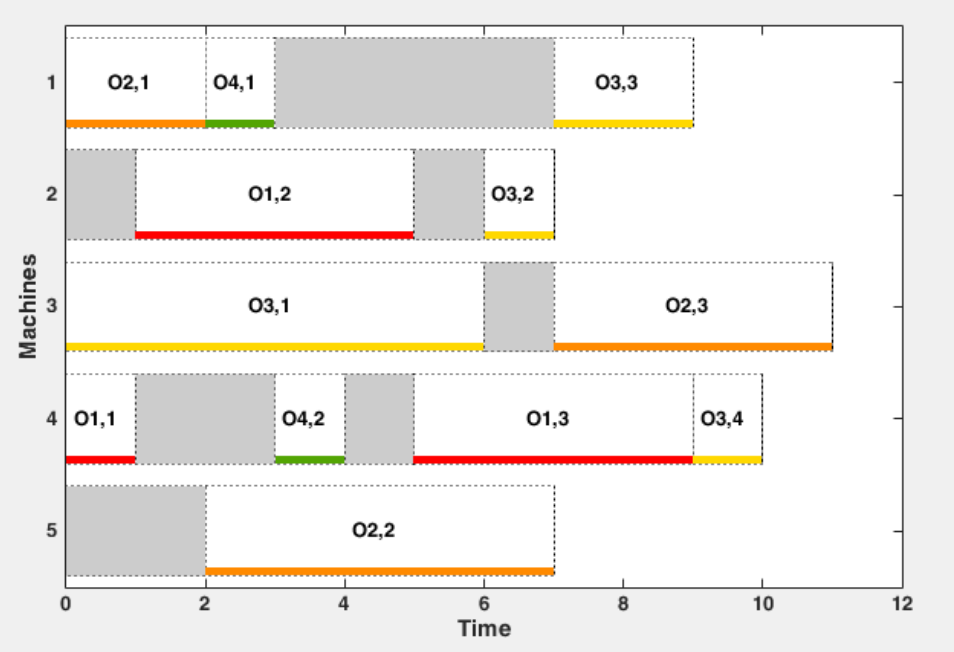

Figura 15: Gráfico de Gantt do cenário 4x5 – PSO + (RRHC, SA e TS).

5.1.2 Cenário 8x8

Este cenário é o único do tipo parcialmente flexível contido nessa base de dados, gozando assim do mecanismo de tratamento de não utilização de recursos produtivos detentores de tempos nulos, diferente da abordagem realizada por (KACEM; HAMMADI; BORNE, 2002). Contém 27 operações, 8 *jobs* e 8 recursos produtivos, os tempos estão apresentados na Tabela 5.

Tabela 5: Cenário 8x8 extraído de KACEM; HAMMADI; BORNE (2002).

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
1	$O_{1,1}$	5	3	5	3	3	-	10	9
	$O_{1,2}$	10	-	5	8	3	9	9	6
	$O_{1,3}$	-	10	-	5	6	2	4	5
2	$O_{2,1}$	5	7	3	9	8	-	9	-
	$O_{2,2}$	-	8	5	2	6	7	10	9
	$O_{2,3}$	-	10	-	5	6	4	1	7
	$O_{2,4}$	10	8	9	6	4	7	-	-
3	$O_{3,1}$	10	-	-	7	6	5	2	4
	$O_{3,2}$	-	10	6	4	8	9	10	-
	$O_{3,3}$	1	4	5	6	-	10	-	7
4	$O_{4,1}$	3	1	6	5	9	7	8	4
	$O_{4,2}$	12	11	7	8	10	5	6	9
	$O_{4,3}$	4	6	2	10	3	9	5	7
5	$O_{5,1}$	3	6	7	8	9	-	10	-
	$O_{5,2}$	10	-	7	4	9	8	6	-
	$O_{5,3}$	-	9	8	7	4	2	7	-
	$O_{5,4}$	11	9	-	6	7	5	3	6
6	$O_{6,1}$	6	7	1	4	6	9	-	10
	$O_{6,2}$	11	-	9	9	9	7	6	4
	$O_{6,3}$	10	5	9	10	11	-	10	-
7	$O_{7,1}$	5	4	2	6	7	-	10	-
	$O_{7,2}$	-	9	-	9	11	9	10	5
	$O_{7,3}$	-	8	9	3	8	6	-	10
8	$O_{8,1}$	2	8	5	9	-	4	-	10
	$O_{8,2}$	7	4	7	8	9	-	10	-
	$O_{8,3}$	9	9	-	8	5	6	7	1
	$O_{8,4}$	9	-	3	7	1	5	8	-

Os parâmetros testados neste cenário foram: Número de iterações PSO = 15; Número de partículas = 20. Também foi alcançado pelas três meta-heurísticas auxiliares o ótimo encontrado até o presente momento que é, $Ck = 14$, $Wm = 12$ e $Wt = 77$. O resultado do Gráfico de Gantt pode ser observado na Figura 16. No

quesito tempo, o PSO + RRHC teve o melhor resultado, levando 2.1 minutos para executar, o PSO + SA 2.2 minutos e o PSO + TS 2.5 minutos, o gráfico de convergência pode ser visto na Figura 17, cujo menor valor de aptidão para a soma ponderada é de 19,45, visto que $14 * 0{,}8 + 12 * 0{,}11 + 77 * 0{,}09 = 19{,}45$. As Figuras 18, 19 e 20, mostram o gráfico *boxplot* dos valores de aptidão ao longo das iterações.

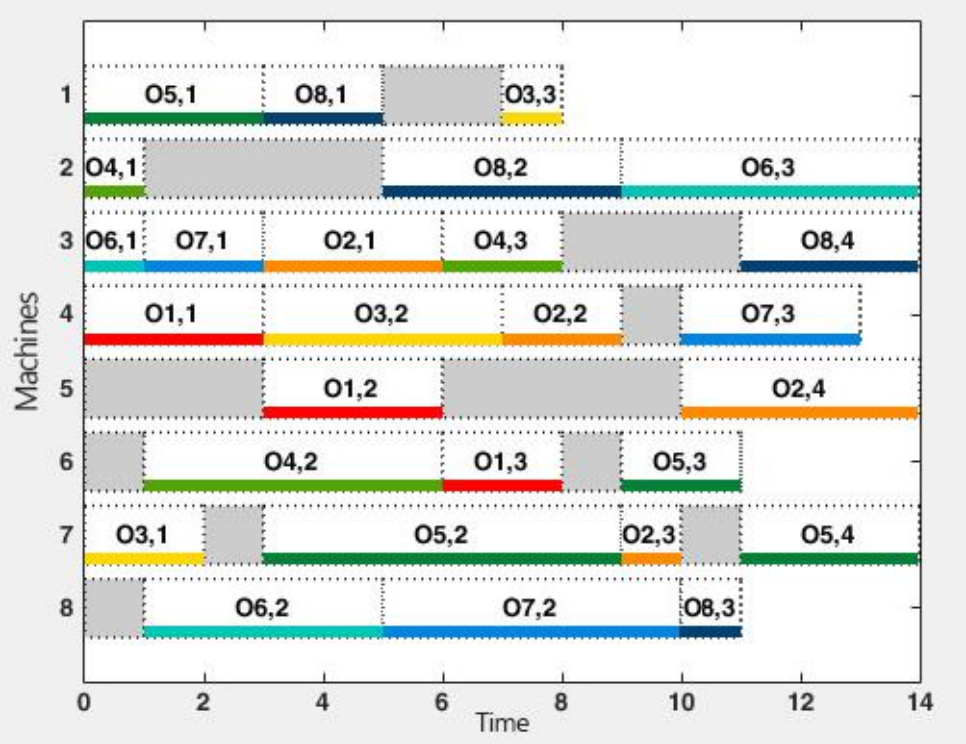

Figura 16: Gráfico de Gantt do cenário 8x8 – PSO + (RRHC, SA e TS).

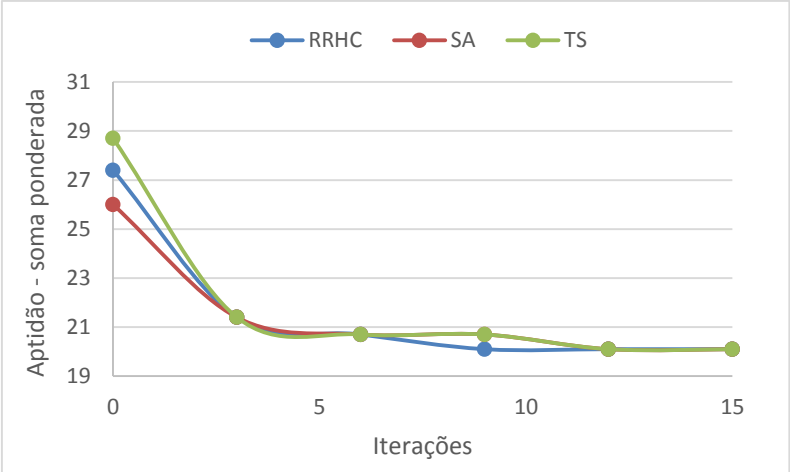

Figura 17: Gráfico de convergência do cenário 8x8 da melhor solução.

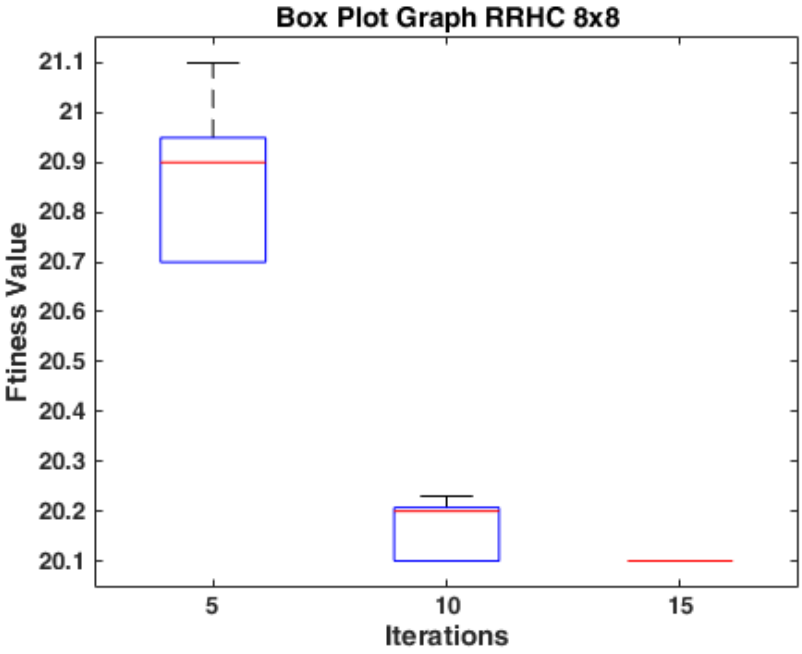

Figura 18: *Boxplot* RRHC para o cenário 8x8

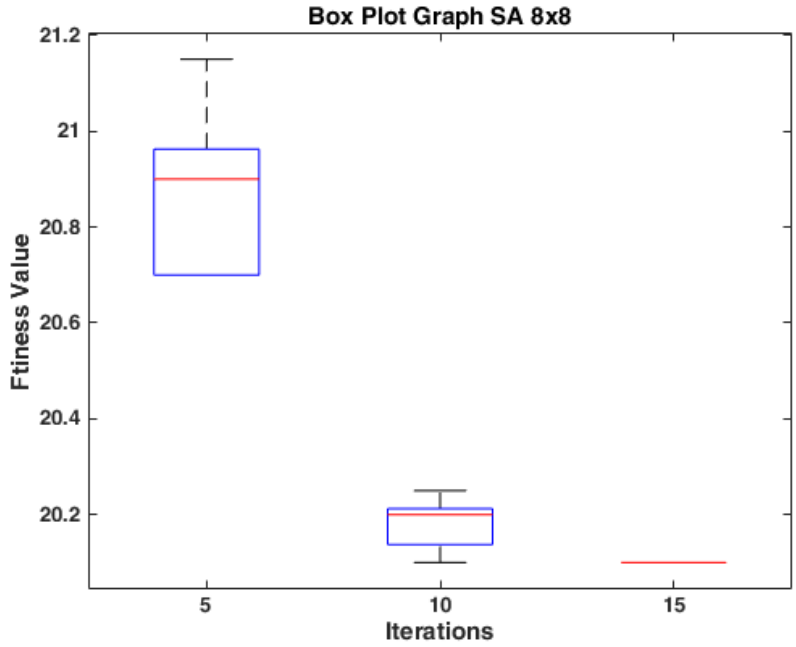

Figura 19: *Boxplot* SA para o cenário 8x8

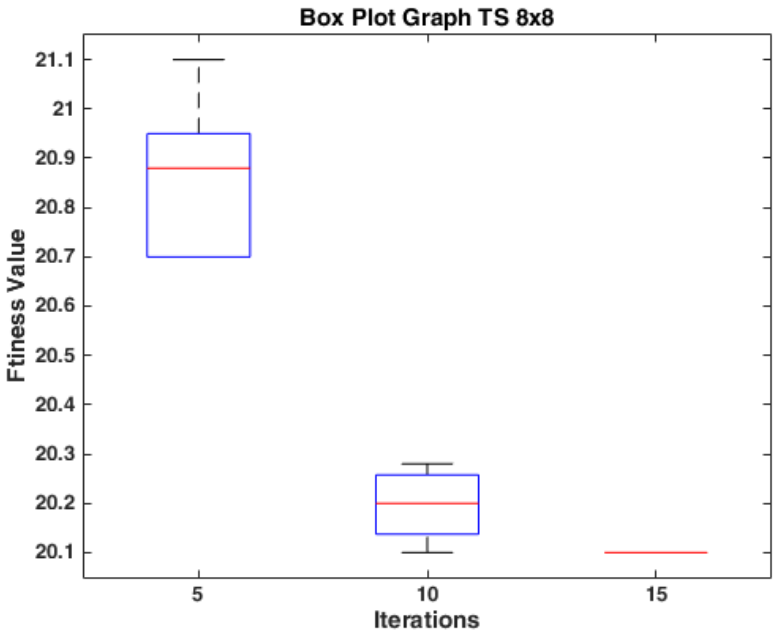

Figura 20: *Boxplot* TS para o cenário 8x8

5.1.3 Cenário 10x10

O cenário atual apresenta um pouco mais de complexidade por ser maior e totalmente flexível, o que aumenta assim a quantidade de soluções factíveis. Este cenário possui 30 operações, 10 *jobs* que são processados em 10 recursos produtivos. Os tempos de processamento de cada recurso estão presentes na Tabela 6.

Tabela 6: Cenário 10x10 extraído de KACEM; HAMMADI; BORNE (2002).

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	5
	$O_{1,2}$	4	1	1	3	4	8	10	4	11	4
	$O_{1,3}$	3	2	5	1	5	6	9	5	10	3
2	$O_{2,1}$	2	10	4	5	9	8	4	15	8	4
	$O_{2,2}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,3}$	6	11	2	7	5	3	5	14	9	2
3	$O_{3,1}$	8	5	8	9	4	3	5	3	8	1
	$O_{3,2}$	9	3	6	1	2	6	4	1	7	2
	$O_{3,3}$	7	1	8	5	4	9	1	2	3	4
4	$O_{4,1}$	5	10	6	4	9	5	1	7	1	6
	$O_{4,2}$	4	2	3	8	7	4	6	9	8	4
	$O_{4,3}$	7	3	12	1	6	5	8	3	5	2
5	$O_{5,1}$	7	10	4	5	6	3	5	15	2	6
	$O_{5,2}$	5	6	3	9	8	2	8	6	1	7
	$O_{5,3}$	6	1	4	1	10	4	3	11	13	9
6	$O_{6,1}$	8	9	10	8	4	2	7	8	3	10
	$O_{6,2}$	7	3	12	5	4	3	6	9	2	15
	$O_{6,3}$	4	7	3	6	3	4	1	5	1	11
7	$O_{7,1}$	1	7	8	3	4	9	4	13	10	7
	$O_{7,2}$	3	8	1	2	3	6	11	2	13	3
	$O_{7,3}$	5	4	2	1	2	1	8	14	5	7
8	$O_{8,1}$	5	7	11	3	2	9	8	5	12	8
	$O_{8,2}$	8	3	10	7	5	13	4	6	8	4
	$O_{8,3}$	6	2	13	5	4	3	5	7	9	5
9	$O_{9,1}$	3	9	1	3	8	1	6	7	5	4
	$O_{9,2}$	4	6	2	5	7	3	1	9	6	7
	$O_{9,3}$	8	5	4	8	6	1	2	3	10	12
10	$O_{10,1}$	4	3	1	6	7	1	2	6	20	6
	$O_{10,2}$	3	1	8	1	9	4	1	4	17	15
	$O_{10,3}$	9	2	4	2	3	5	2	4	10	23

Com o aumento da complexidade houve também um aumento na quantidade de partículas para que o ótimo fosse atingido. Número de partículas = 30, mantendo as 15 iterações do PSO. A melhor solução encontrada, foi a mesma nas três meta-heurísticas, $Ck = 7$, $Wm = 5$ e $Wt = 43$, igual a encontrada dos melhores trabalhos citados no Capítulo 3. A programação para este resultado está no Gráfico de Gantt da Figura 21. O PSO + RRHC possui o tempo de execução de 36 minutos, PSO + SA obteve o tempo de 40 minutos, e o PSO + TS finalizou a execução em 53 minutos. É possível ver uma variância um pouco maior no gráfico de convergência apresentado pela Figura 22, com vantagem do RRHC, que convergiu primeiro para o resultado da soma ponderada, 10,02, dado que $7 * 0{,}8 + 5 * 0{,}11 + 43 * 0{,}09 = 10{,}02$. Constam nas Figuras 23, 24 e 25 os gráficos *boxplots* para este cenário.

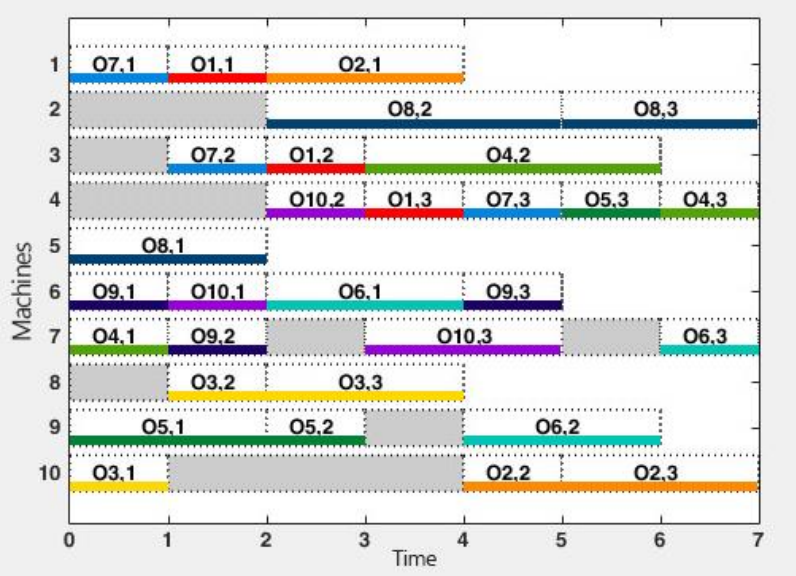

Figura 21: Gráfico de Gantt do cenário 10x10 – PSO + (RRHC, SA e TS).

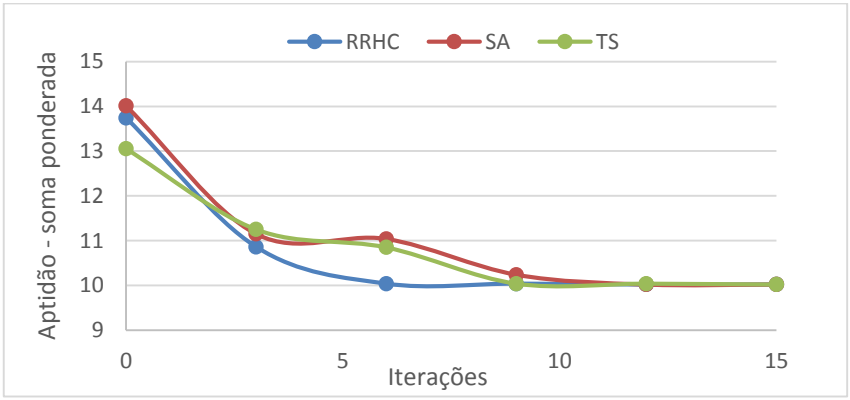

Figura 22: Gráfico de convergência do cenário 10x10.

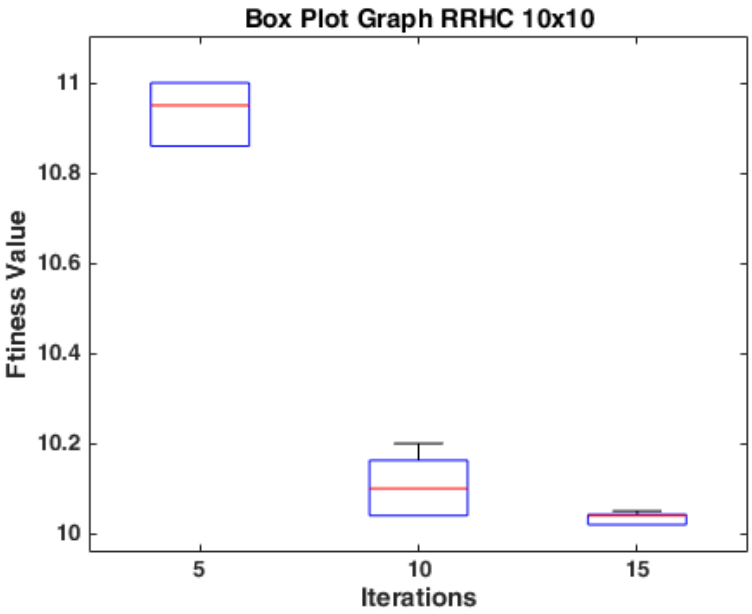

Figura 23: *Boxplot* RRHC para o cenário 10x10

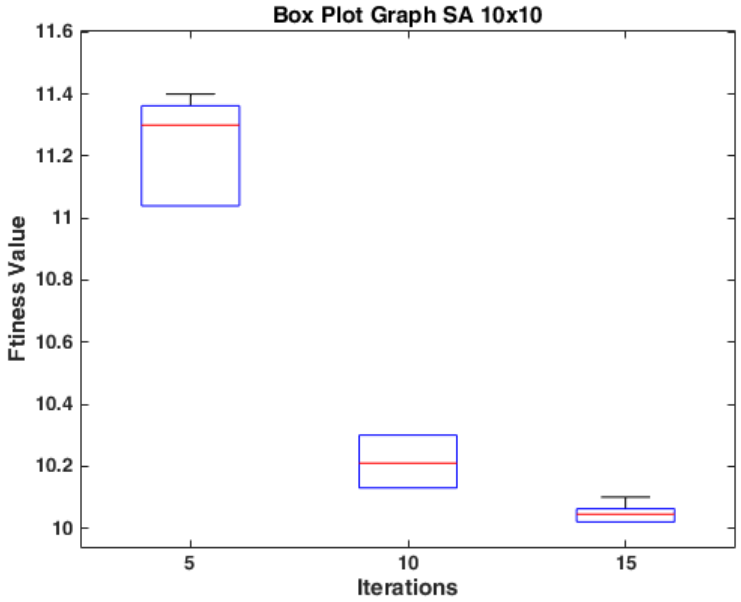

Figura 24: *Boxplot* SA para o cenário 10x10

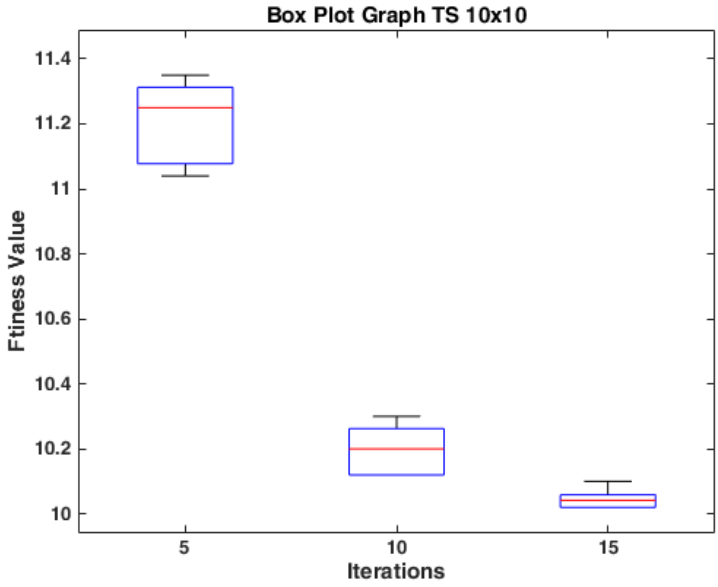

Figura 25: *Boxplot* TS para o cenário 10x10

5.1.4 Cenário 15x10

Para finalizar os testes preliminares encontra-se o cenário mais complexo estudado, este cenário é do tipo totalmente flexível, possui 56 operações distribuídas em 15 *jobs*, e estes precisam ser processados em 10 recursos. Seus tempos de processamento são exibidos na Tabela 7.

Tabela 7: Cenário 15x10 extraído de KACEM; HAMMADI; BORNE (2002).

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	4
	$O_{1,2}$	1	1	3	4	8	10	4	11	4	3
	$O_{1,3}$	2	5	1	5	6	9	5	10	3	2
	$O_{1,4}$	10	4	5	9	8	4	15	8	4	4
2	$O_{2,1}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,2}$	6	11	2	7	5	3	5	14	9	2
	$O_{2,3}$	8	5	8	9	4	3	5	3	8	1
	$O_{2,4}$	9	3	6	1	2	6	4	1	7	2
3	$O_{3,1}$	7	1	8	5	4	9	1	2	3	4
	$O_{3,2}$	5	10	6	4	9	5	1	7	1	6
	$O_{3,3}$	4	2	3	8	7	4	6	9	8	4
	$O_{3,4}$	7	3	12	1	6	5	8	3	5	2
4	$O_{4,1}$	6	2	5	4	1	2	3	6	5	4
	$O_{4,2}$	8	5	7	4	1	2	36	5	8	5
	$O_{4,3}$	9	6	2	4	5	1	3	6	5	2
	$O_{4,4}$	11	4	5	6	2	7	5	4	2	1
5	$O_{5,1}$	6	9	2	3	5	8	7	4	1	2
	$O_{5,2}$	5	4	6	3	5	2	28	7	4	5
	$O_{5,3}$	6	2	4	3	6	5	2	4	7	9
	$O_{5,4}$	6	5	4	2	3	2	5	4	7	5
6	$O_{6,1}$	4	1	3	2	6	9	8	5	4	2
	$O_{6,2}$	1	3	6	5	4	7	5	4	6	5
7	$O_{7,1}$	1	4	2	5	3	6	9	8	5	4
	$O_{7,2}$	2	1	4	5	2	3	5	4	2	5
8	$O_{8,1}$	2	3	6	2	5	4	1	5	8	7
	$O_{8,2}$	4	5	6	2	3	5	4	1	2	5
	$O_{8,3}$	3	5	4	2	5	49	8	5	4	5
	$O_{8,4}$	1	2	36	5	2	3	6	4	11	2
9	$O_{9,1}$	6	3	2	22	44	11	10	23	5	1
	$O_{9,2}$	2	3	2	12	15	10	12	14	18	16
	$O_{9,3}$	20	17	12	5	9	6	4	7	5	6
	$O_{9,4}$	9	8	7	4	5	8	7	4	56	2
10	$O_{10,1}$	5	8	7	4	56	3	2	5	4	1
	$O_{10,2}$	2	5	6	9	8	5	4	2	5	4
	$O_{10,3}$	6	3	2	5	4	7	4	5	2	1
	$O_{10,4}$	3	2	5	6	5	8	7	4	5	2
11	$O_{11,1}$	1	2	3	6	5	2	1	4	2	1
	$O_{11,2}$	2	3	6	3	2	1	4	10	12	1
	$O_{11,3}$	3	6	2	5	8	4	6	3	2	5
	$O_{11,4}$	4	1	45	6	2	4	1	25	2	4
12	$O_{12,1}$	9	8	5	6	3	6	5	2	4	2
	$O_{12,2}$	5	8	9	5	4	75	63	6	5	21
	$O_{12,3}$	12	5	4	6	3	2	5	4	2	5
	$O_{12,4}$	8	7	9	5	6	3	2	5	8	4
13	$O_{13,1}$	4	2	5	6	8	5	6	4	6	2
	$O_{13,2}$	3	5	4	7	5	8	6	6	3	2
	$O_{13,3}$	5	4	5	8	5	4	6	5	4	2
	$O_{13,4}$	3	2	5	6	5	4	8	5	6	4
14	$O_{14,1}$	2	3	5	4	6	5	4	85	4	5
	$O_{14,2}$	6	2	4	5	8	6	5	4	2	6
	$O_{14,3}$	3	25	4	8	5	6	3	2	5	4
	$O_{14,4}$	8	5	6	4	2	3	6	8	5	4
15	$O_{15,1}$	2	5	6	8	5	6	3	2	5	4
	$O_{15,2}$	5	6	2	5	4	2	5	3	2	5
	$O_{15,3}$	4	5	2	3	5	2	8	4	7	5
	$O_{15,4}$	6	2	11	14	2	3	6	5	4	8

Neste cenário, além do aumento nos parâmetros do PSO, também foi necessário uma alteração nos parâmetros de cada meta-heurística, dado que comparado ao cenário anterior, este possui 26 operações a mais. Foi executado este cenário com os parâmetros: Número de iterações PSO = 30, mantendo as 30 partículas. Já nas meta-heurísticas, o $\alpha = 0.90$ (SA), iterações = 1000 (TS), e iterações = 3000 (RRHC). A melhor solução foi encontrada somente pelos algoritmos RRHC e SA, cujos valores ficaram em: $Ck = 11$, $Wm = 11$ e $Wt = 91$, enquanto o TS obteve $Ck = 11$, $Wm = 11$ e $Wt = 93$. O Gráfico de Gantt da Figura 26 ilustra como foi resolvido o subproblema de programação pelo PSO + RRHC e PSO + SA. Os tempos de compilação foram: PSO + RRHC 75 minutos, PSO + SA 88 minutos e PSO + TS 129 minutos. O gráfico de convergência da Figura 27 elucida também uma ligeira vantagem do algoritmo RRHC, pois atingiu primeiro o menor valor da função de aptidão, 18,2, resultante da soma ponderada $11 * 0{,}8 + 11 * 0{,}11 + 91 * 0{,}09 = 18{,}2$. Foram dispostos nas Figuras 28, 29 e 30 os gráficos boxplots para o PSO + RRHC, PSO + SA e PSO + TS respectivamente.

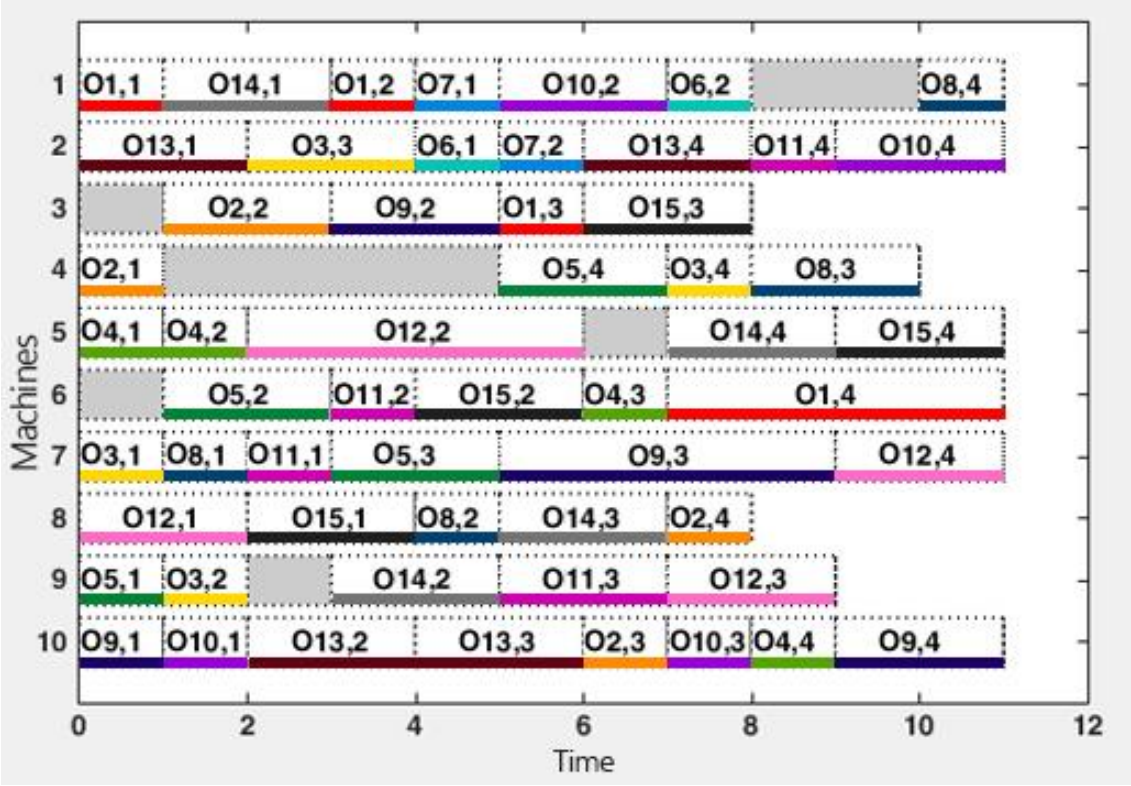

Figura 26: Gráfico de Gantt do cenário 15x10 – PSO + (RRHC e SA).

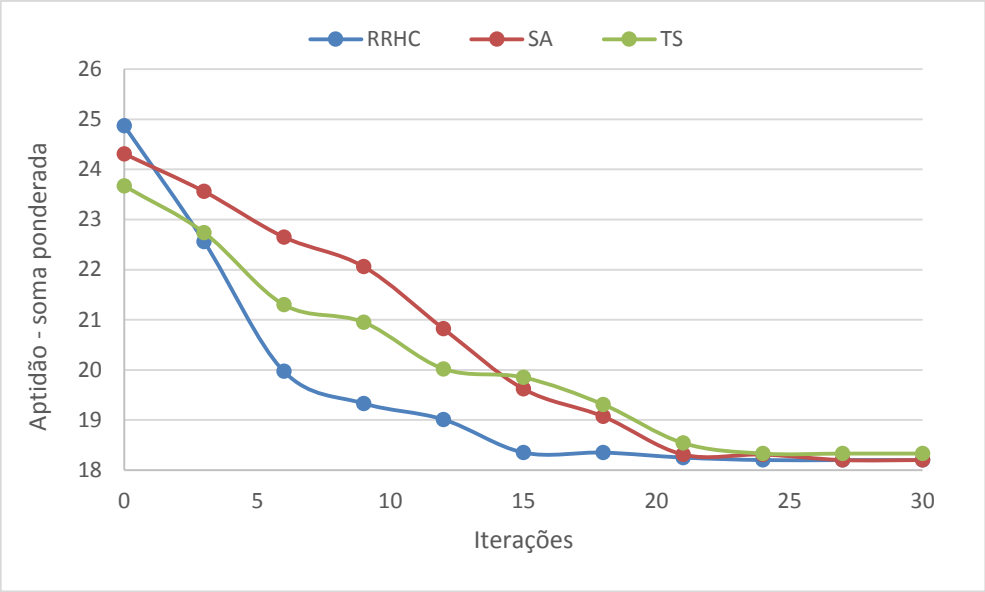

Figura 27: Gráfico de convergência do cenário 15x10.

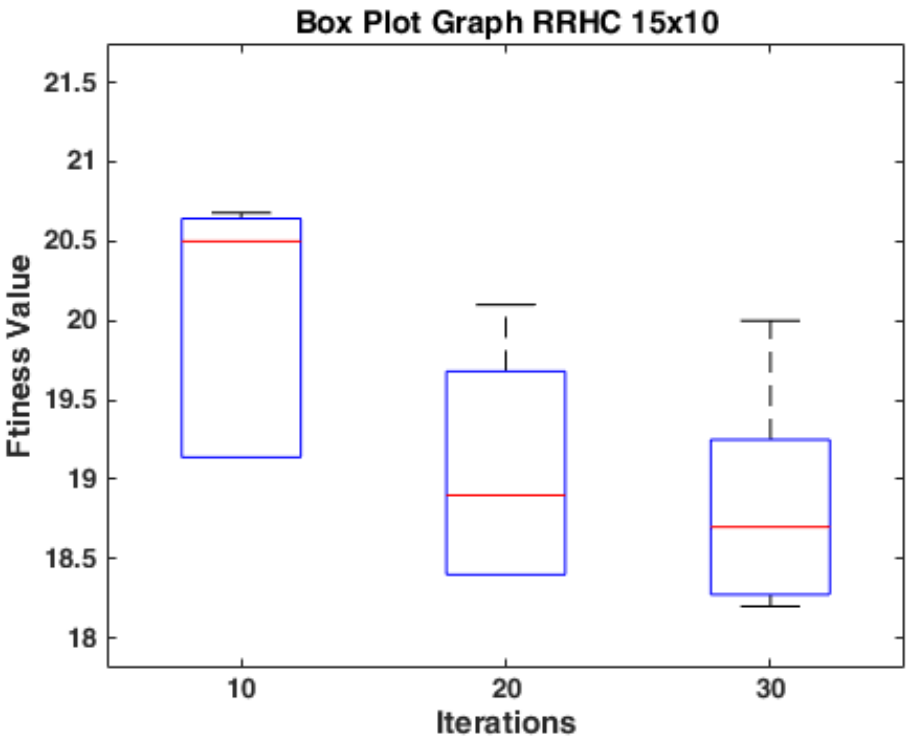

Figura 28: *Boxplot* RRHC para o cenário 15x10

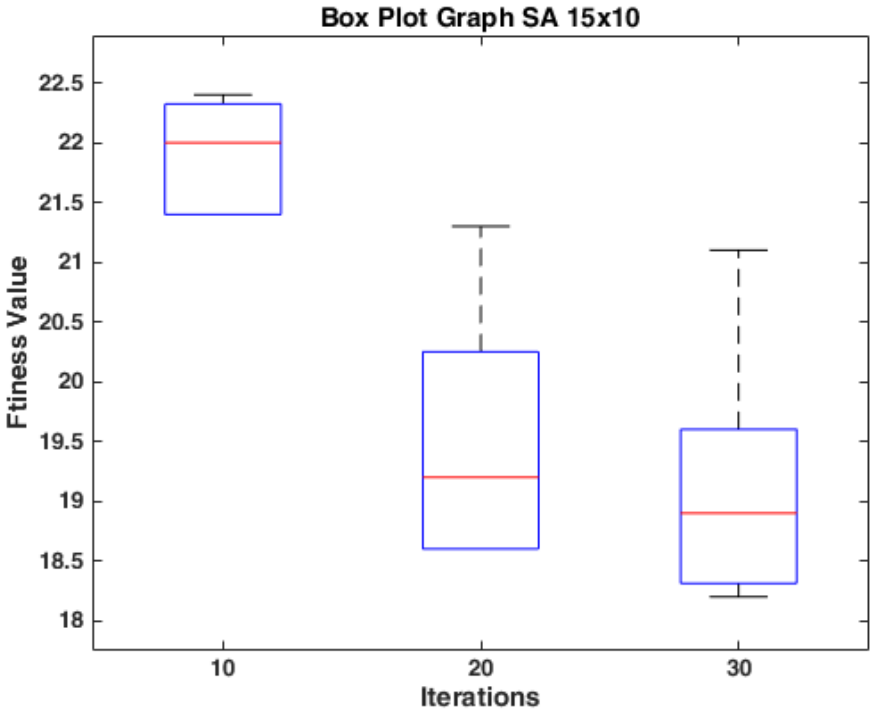

Figura 29: *Boxplot* SA para o cenário 15x10

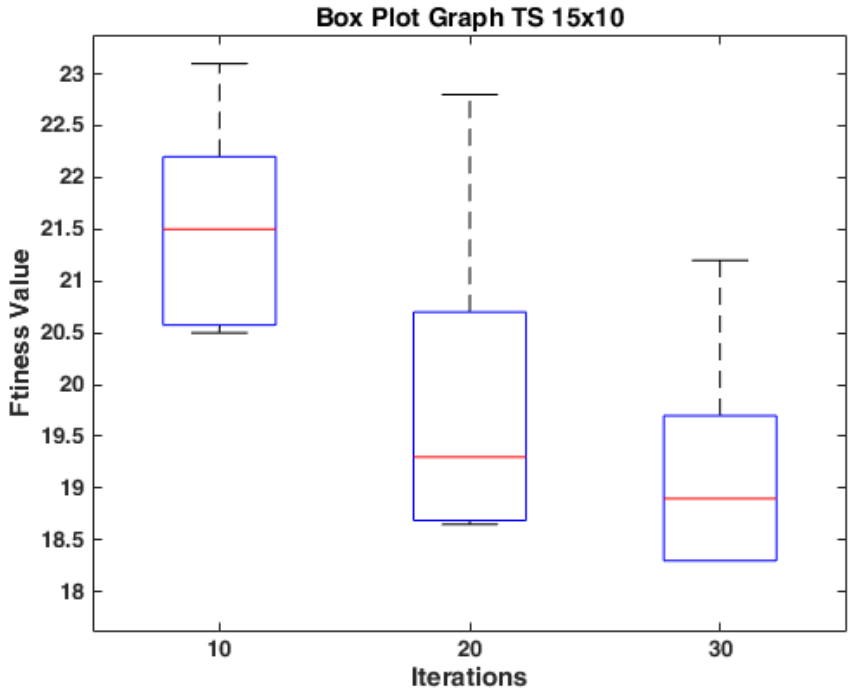

Figura 30: *Boxplot* TS para o cenário 15x10

5.2 Comparação de resultados

Este trabalho possui dois objetos de comparação, um é do algoritmo híbrido com a utilização das três meta-heurísticas auxiliares, comparado a outros trabalhos similares encontrados na literatura, ilustrado na Tabela 8. O outro objeto de comparação é do desempenho em tempo de execução em relação a um computador pessoal (3.3GHz, 6 núcleos e 8GB de RAM), de três algoritmos de busca local para a resolução do subproblema de programação, encontrado na Tabela 9.

A comparação do tempo de execução da hibridização com estudos encontrados na literatura não foi o foco dessa pesquisa pois foi há uma divergência nas ferramentas usadas, uma vez que para essa pesquisa foi optado pelo MATLAB, com o adendo de que a implementação foi conduzida de forma paralelizada, ou seja, as partículas foram processadas paralelamente nos 6 núcleos dispostos.

Para melhor compreensão, as abreviaturas do comparativo situado na Tabela 8, competem as seguintes pesquisas: AL + CGA, KACEM; HAMMADI; BORNE (2002); PSO + SA, XIA; WU (2005); PSO + TS, ZHANG et al. (2009); MOPSO, MOSLEHI; MAHNAM (2011); hDPSO, SHAO et al. (2013); e Dist PSO, NOUIRI et al. (2015).

Tabela 8: Comparativo de resultados dos trabalhos usando a base de dados de (Kacem et al., 2002a, 2002b).

nxm	Objetivo	AL + CGA	PSO + SA	PSO + TS	MOPSO	hDPSO	Dist PSO	PSO		
								RRHC	SA	TS
8x8	C_m	15	15	14	14	14	17	14	14	14
	W_m	-	12	12	12	12	-	12	12	12
	W_t	75	75	77	77	72	-	77	77	77
10x10	C_m	7	7	7	7	7	8	7	7	7
	W_m	5	6	6	5	5	-	5	5	5
	W_t	41	44	43	43	43	-	43	43	43
15x10	C_m	23	12	11	11	11	-	11	11	11
	W_m	11	11	11	11	11	-	11	11	11
	W_t	91	91	93	91	91	-	91	91	92
Max: Iterações - População	-	500 - 100	100 - 50	100 - 50	100 -100	100***** - 500	200 - 100	30 - 30		

*****valor aproximado

Como visto na Tabela 8, a hibridização do PSO com as meta-heurísticas auxiliares atingiu as soluções ótimas encontradas até o presente momento, salvo pela combinação PSO + TS que no objetivo Wt do cenário 15x10 obteve 92 e não 91 como as outras hibridizações. Contudo o algoritmo proposto apresentou uma melhoria considerável na quantidade de iterações e tamanho da população comparado as outras pesquisa. O MOPSO (MOSLEHI; MAHNAM, 2011), foi o algoritmo referenciado que atingiu os resultados ótimos com menos iterações e partículas em sua população, usando no máximo 100 iterações e 100 partículas, enquanto o algoritmo híbrido proposto utilizou 30 e 30, respectivamente.

A Tabela 9 ratifica a vantagem da meta-heurística auxiliar RRHC com referência ao tempo de execução. Não obstante, a variação encontrada pode ser explanada em virtude dos mecanismos que regem cada algoritmo, o TS que obteve os tempos mais longos, processa cada vizinho encontrado de uma solução corrente, anulando movimentos tabus que, pra este contexto, pode prejudicar a convergência para a solução ótima, demandando assim mais iterações. Por sua vez, o algoritmo SA, fundamentado exclusivamente em sua aleatoriedade necessita de mais repetições em relação ao RRHC, pois sua abordagem de aceitar soluções piores com mais frequência no início do algoritmo o força a explorar de forma mais abrangente o espaço de busca, todavia para a base de dados utilizada, este comportamento resultou em maior demanda de repetições e consequentemente em um maior tempo de execução.

De modo semelhante ao aludido acima sobre o algoritmo SA, opera o RRHC, porém a aleatoriedade se dá ao reiniciar seu procedimento com outra solução inicial, dada uma quantidade de iterações sem melhoria, buscando uma solução ótima de modo ‘guloso’, ou seja, apenas considerando as soluções que demonstraram progresso, o que foi oportuno para o subproblema proposto.

Tabela 9: Comparativo de tempo de execução das meta-heurísticas auxiliares usadas.

nxm	8x8	10x10	15x10
HC	2,1 min.	36 min.	75 min.
SA	2,2 min.	40 min.	88 min.
TS	2,5 min.	53 min.	129 min.

Com exceção do cenário 4x5 que é considerado de pequena escala, foi calculada a média e o desvio padrão para cada cenário, vide Tabela 10, sendo que

cada algoritmo foi executado 10 vezes por cenário, totalizando 120 execuções. Na Tabela 10 pode-se observar que no cenário 8x8, o desvio padrão de todas as meta-heurísticas corresponde a 0, isto é, as hibridizações asseguram a melhor solução em todas as execuções. Ao passo que nos cenários 10x10 e 15x10 os maiores desvios padrão foram de 1,89 e 2,36, respectivamente, e ambos valores representam o objetivo Wt, que é a soma de todos os tempos de processamento das operações, enquanto que para o objetivo Cm (*makespan*), o desvio padrão foi de no máximo 1,44. Deste modo constata-se a assertividade do algoritmo proposto, que é refletida na baixa discrepância dos resultados obtidos para qualquer execução utilizando qualquer meta-heurística.

Tabela 10: Comparativo de resultados mínimos, médios e desvios padrão.

nxm	Objetivo	PSO + RRHC	PSO + SA	PSO + TS	PSO + RRHC	PSO + SA	PSO + TS	PSO + RRHC	PSO + SA	PSO + TS
		Mínimo			Média			Desvio Padrão		
8x8	C_m	14	14	14	14	14	14	0	0	0
	W_m	12	12	12	12	12	12	0	0	0
	W_t	77	77	77	77	77	77	0	0	0
10x10	C_m	7	7	7	7,7	7,8	7,8	1,44	1,3	1,3
	W_m	5	5	5	5,7	5,7	5,8	1,44	1,44	1,3
	W_t	43	43	43	42,2	42,1	42,2	1,89	1,73	1,26
15x10	C_m	11	11	11	11,8	11,8	12,1	1,26	1,26	0,94
	W_m	11	11	11	10,8	10,9	10,5	1,26	1,37	1,58
	W_t	91	91	92	91,6	91,8	92,5	2,09	2,36	1,58

Capítulo 6

CONSIDERAÇÕES FINAIS

Esta pesquisa se faz relevante pelo crescente interesse da comunidade cientifica em métodos para resolução de problemas de otimização, como aqueles classificados em NP-*hard*. O âmago da busca é por soluções ótimas, que possam ser encontradas em tempo hábil, para que sejam aplicadas em problemas encontrados em nosso cotidiano, tal como o usado nesta pesquisa, o *Flexible Job-shop Scheduling Problem* (FJSP). Diante disso, uma gama de propostas para a resolução do FJSP pode ser encontrada na literatura, como modelos elaborados por regras de despacho, algoritmos evolutivos, entre outros. Entretanto é observado uma convergência para soluções baseadas em algoritmos de inteligência coletiva, mais comumente chamados de inteligência de enxame.

Foi proposto um algoritmo de Enxame de Partículas integrado com três meta-heurísticas auxiliares, Arrefecimento Simulado (SA), Busca Tabu (TS) e Subida de Colina Com Reinício Aleatório (RRHC) para a resolução do FJSP*, c*orroborado pela eficiência constatada de algoritmos com características de enxame para a solução subproblema de roteamento, somado à conveniente atribuição do subproblema de programação para um algoritmo de busca local.

Em referência ao subproblema de programação, foi feita uma comparação entre as três meta-heurísticas usando as mesma características, como, mesma função de vizinhança, função de aptidão e base de testes. Ficando claro nessa comparação, a superioridade do algoritmo RRHC perante os outros algoritmos, ratificada pelos testes apresentados na seção anterior.

A favorável aplicabilidade do RRHC, mesmo sendo um algoritmo ‘guloso’ e tecnicamente simples, pode ser atribuída aos procedimentos de melhora impostos nessa pesquisa. Como a inicialização estocástica da solução de programação, uma vez que a cada reinício do HC resultante de uma estagnação de uma solução em um ótimo local, o algoritmo redirecionava essa solução no espaço de busca para um novo ponto inicial, com proximidade a um outro ótimo, que pode ser o global.

Deve-se enfatizar que a simples função de vizinhança foi aperfeiçoada com o método de só movimentar operações em um caminho crítico, em pares. Estes procedimentos renderam ao RRHC uma ligeira vantagem sobre os outros algoritmos, provando sua aptidão como algoritmo para resolução do subproblema de programação de um FJSP.

Com relação a conjuntura do algoritmo, é importante notar que as hibridizações propostas apresentaram uma solução com um número menor de iterações e população considerável comparado às outras pesquisas, refletindo também os aprimoramentos do algoritmo PSO. Isso ocorreu devido ao desmembramento de uma solução baseada em níveis de operação para baseado em operações, usando as variações de partículas caso haja múltiplas possibilidades de recursos para as operações, e as processando paralelamente (implementação possível no MATLAB). E dos registros que garantem o processamento único de cada partícula.

Resultante das não numerosas iterações e população, foi possível mensurar com mais facilidade a relevância das variáveis e divisão do problema, assim constatando que as experiências cognitivas e sociais (pB e gB) devem influenciar de forma igualitária, havendo um peso ($weight$) suficiente dessas experiências para explorar soluções vizinhas não tão próximas, resultando assim na aquisição do roteamento ótimo. Todavia, obter um roteamento ótimo não garante uma solução ótima, as variáveis dos algoritmos auxiliares devem garantir a melhor programação, visto que as partículas não são repetidas, logo, suas iterações são maiores.

Quanto ao desempenho do algoritmo em relação as diferentes execuções de cada problema, o cálculo de média e desvio padrão, somado aos gráficos *boxplot* mostram a eficiência do modelo híbrido proposto. Cujo modelo, garante que dada a execução de qualquer cenário da base de dados proposta, utilizando uma das meta-heurísticas auxiliar abordadas, irá produzir uma solução que distinguirá em no máximo 1.87 unidades de tempo do ótimo obtido até a presente data – como mostra a Tabela 10. E sendo assim, unindo este fator com o restante, é possível verificar que os objetivos estabelecidos foram satisfatoriamente atendidos e legitimados pelos testes utilizados.

Afirma-se por conseguinte que esta pesquisa apresenta uma contribuição relevante em forma de comparativo para resolução do FJSP, indicando procedimentos inovadores para a hibridização e o algoritmo RRHC para a resolução do subproblema de programação.

6.1 Trabalhos Futuros

Alguns trabalhos adicionais são factíveis a fim de aprimorar o tempo computacional, que nessa pesquisa foi secundário. Além disso, é possível aumentar a complexidade, impondo condições para manutenção, tratamento para quebra de máquinas ou abstenções, ou até mesmo objetivos como o *tardiness* (PINEDO, 2016). Para checar a assertividade do RRHC, é sugerido também sua aplicação com outro algoritmo, como Colônia de Formigas e Colônia Artificial de Abelhas. Além disso, a utilização de outros cenários com escalas diferentes é sugerido para o modelo híbrido proposto, por apresentarem características diferentes, evidenciando melhor a solidez do algoritmo.

Pode-se também utilizar uma metodologia alternativa nos pesos dos objetivos que regem FJSP, optando por inicializar as partículas apenas com soluções boas de objetivos diferentes, com a finalidade de explorar soluções próximas que contenham os melhores valores para todos os objetivos estipulados.

REFERÊNCIAS

BAGHERI, A. et al. An artificial immune algorithm for the flexible job-shop scheduling problem. **Future Generation Computer Systems**, v. 26, n. 4, p. 533–541, 2010.

BASHIR, Z. A.; EL-HAWARY, M. E. Applying Wavelets to Short-Term Load Forecasting Using PSO-Based Neural Networks. **IEEE Transactions on Power Systems**, v. 24, n. 1, p. 20–27, fev. 2009.

BRANDIMARTE, P. Routing and scheduling in a flexible job shop by tabu search. **Annals of Operations Research**, v. 41, n. 3, p. 157–183, 1993.

ČERNÝ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. **Journal of Optimization Theory and Applications**, v. 45, n. 1, p. 41–51, 1985.

CHEN, B.-R.; FENG, X.-T. Particle swarm optimization with contracted ranges of both search space and velocity. **Journal of Northeastern University**, v. 26, n. 5, p. 488–491, 2005.

COELLO, C. A Short Tutorial on Evolutionary Multiobjective Optimization. **Evolutionary Multi-Criterion Optimization**, v. 1993, p. 21–40, 2001.

COELLO, C. A.; REYES-SIERRA, M. Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. **International Journal of Computational Intelligence Research**, v. 2, n. 3, p. 287–308, 2006.

COLLETTE, Y.; SIARRY, P. Optimisation multiobjectif. In: **Optimisation multiobjectif**. Paris: Edition Eyrolles, 2002. p. 47–80.

DEB, K. **Multi-objective optimization using evolutionary algorithms**. Chichester, UK: John Wiley & Sons, 2001. v. 16

DORIGO, M.; BLUM, C. Ant colony optimization theory: A survey. **Theoretical Computer Science**, v. 344, n. 2-3, p. 243–278, 2005.

EBERHART, R. C.; SHI, Y. **Comparing inertia weights and constriction factors in particle swarm optimization**Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512). **Anais**...IEEE, 2000Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=870279>

EL MARAGHY, H. A. Flexible and reconfigurable manufacturing systems paradigms. **Flexible Services and Manufacturing Journal**, v. 17, n. 4 SPECIAL ISSUE, p. 261–276, 2006.

GANDOMI, A. H. et al. Chaos-enhanced accelerated particle swarm optimization. **Communications in Nonlinear Science and Numerical Simulation**, v. 18, n. 2, p. 327–340, 2013.

GAREY, M. R.; JOHNSON, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). **Computers and Intractability**, p. 340, 1979.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The Complexity of Flowshop and Jobshop Scheduling. **Mathematics of Operations Research**, v. 1, n. 2, p. 117–129, maio 1976.

GLOVER, F. Tabu Search-Part I. **ORSA Journal on Computing**, v. 1, n. 3, p. 190–206, 1889.

GLOVER, F. **Tabu Search: A tutorialInterfaces**, 1990.

GROSAN, C. et al. Hybrid Particle Swarm – Evolutionary Algorithm for Search and Optimization. In: **MICAI 2005: Advances in Artificial Intelligence**. Monterrey, Mexico: Springer Berlin Heidelberg, 2005. p. 623–632.

HAHN, B.; VALENTINE, D. **Essential MATLAB for engineers and scientists**. Fifth Edit ed.[s.l.] Academic Press, 2013.

HART, E.; ROSS, P.; CORNE, D. Evolutionary scheduling: A review. **Genetic Programming and Evolvable Machines**, v. 6, n. 2, p. 191–220, 2005.

HEINONEN, J.; PETTERSSON, F. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. **Applied Mathematics and Computation**, v. 187, n. 2, p. 989–998, 2007.

HONGBO LIU; ABRAHAM, A. **Fuzzy adaptive turbulent particle swarm optimization**Fifth International Conference on Hybrid Intelligent Systems (HIS'05). **Anais**...IEEE, 2005Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1587787>

HUANG, S.; TIAN, N.; JI, Z. Particle swarm optimization with variable neighborhood search for multiobjective flexible job shop scheduling problem. **International Journal of Modeling, Simulation, and Scientific Computing**, v. 7, n. 3, p. 1–17, 2016.

JUANG, C.-F.; WANG, C.-Y. A self-generating fuzzy system with ant and particle swarm cooperative optimization. **Expert Systems with Applications**, v. 36, n. 3, p. 5362–5370, abr. 2009.

KACEM, I.; HAMMADI, S.; BORNE, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. **IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)**, v. 32, n. 1, p. 1–13, fev. 2002.

KATHAWALA, Y.; ALLEN, W. R. Expert Systems and Job Shop Scheduling. **International Journal of Operations & Production Management**, v. 13, n. 2, p. 23–35, fev. 1993.

KENNEDY, J.; EBERHART, R. **Particle swarm optimization**Proceedings of ICNN'95 - International Conference on Neural Networks. **Anais**...IEEE, 1995Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=488968>

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science (New York, N.Y.)**, v. 220, n. 4598, p. 671–680, 1983.

LØVBJERG, M.; RASMUSSEN, T. K. Hybrid Particle Swarm Optimiser with Breeding and Subpopulations. **Proc. 3rd Genetic Evolutionary Computation Conf.**, p. 469 – 476, 2001.

LU, Z.; HOU, Z.; DU, J. Particle Swarm Optimization with Adaptive Mutation. **Frontiers of Electrical and Electronic Engineering in China**, v. 1, n. 1, p. 99–104, jan. 2006.

MASTROLILLI, M.; GAMBARDELLA, L. M. Effective Neighbourhood Functions for the Flexible Job Shop Problem. **Journal of Scheduling**, v. 3, n. 1, p. 3–20, 2000.

METROPOLIS, N. et al. Equation of State Calculations by Fast Computing Machines. **The Journal of Chemical Physics**, v. 21, n. 6, p. 1087, 1953.

MIRANDA, V.; FONSECA, N. EPSO - Evolutionary Particle Swarm Optimization , a New Algorithm with Applications in Power Systems. **Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES**, p. 745–750, 2002.

MONSON, C. K.; SEPPI, K. D. The Kalman Swarm. In: **Genetic and Evolutionary Computation – GECCO 2004**. Seattle, USA: Springer Berlin Heidelberg, 2004. p. 140–150.

MOSLEHI, G.; MAHNAM, M. A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. **International Journal of Production Economics**, v. 129, n. 1, p. 14–22, 2011.

MOSTAGHIM, S.; TEICH, J. **Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO)**Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706). **Anais**...IEEE, 2003Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1202243>

MOTAGHEDI-LARIJANI, A.; SABRI-LAHAIE, K.; HEYDARI, M. Solving Flexible Job Shop Scheduling with Multi Objective Approach. **International …**, v. 21, n. 4, p. 197–209, 2010.

NIE RU; YUE JIANHUA. **A GA and Particle Swarm Optimization based hybrid algorithm**2008 IEEE Congress on Evolutionary Computation (IEEE World Congress

on Computational Intelligence). **Anais**...IEEE, jun. 2008Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4630925>

NOUIRI, M. et al. An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. **Journal of Intelligent Manufacturing**, 5 fev. 2015.

PEI, Z.; HUA, X.; HAN, J. **The Clustering Algorithm Based on Particle Swarm Optimization Algorithm**2008 International Conference on Intelligent Computation Technology and Automation (ICICTA). **Anais**...IEEE, out. 2008Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4659461>

PINEDO, M. L. **Planning and Scheduling in Manufacturing and Services**. 2. ed. New York, NY: Springer New York, 2009.

PINEDO, M. L. **Scheduling**. 4. ed. Boston, MA: Springer US, 2012.

PINEDO, M. L. **Scheduling: Theory, Algorithms, and Systems**. Cham: Springer International Publishing, 2016.

ROSHANAEI, V.; ELMARAGHY, H. A. A. **Mathematical Modelling and Optimization of Flexible Job Shops Scheduling Problem**. [s.l.] University of Windsor, 2012.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3 edition ed.[s.l.] Pearson, 1995. v. 9

SCHONLAU, M.; WELCH, W. J.; JONES, D. R. Global versus local search in constrained optimization of computer models. **Lecture NotesMonograph Series**, v. 34, p. 11–25, 1998.

SHAO, X. et al. Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, v. 67, n. 9-12, p. 2885–2901, 2013.

SHI, Y.; EBERHART, R. A modified particle swarm optimizer. **1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)**, p. 69–73, 1998.

STELTH, P.; ROY, G. LE. Projects ’ Analysis through CPM (Critical Path Method). **School of Doctoral Studies (European Union) Journal**, n. 1, p. 10–51, 2009.

TAY, J. C.; HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. **Computers and Industrial Engineering**, v. 54, n. 3, p. 453–473, 2008.

WEN-JUN ZHANG; XIAO-FENG XIE. **DEPSO: hybrid particle swarm with differential evolution operator**SMC’03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme -

System Security and Assurance (Cat. No.03CH37483). **Anais**...IEEE, 2003Disponível em: <http://link.springer.com/10.1007/978-3-642-17563-3_50>

WICKRAMASINGHE, U.; LI, X. Choosing Leaders for Multi-objective PSO Algorithms Using Differential Evolution. In: **Simulated Evolution and Learning**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 249–258.

XIA, W.; WU, Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. **Computers & Industrial Engineering**, v. 48, n. 2, p. 409–425, 2005.

YAN CHEN; ZENG-ZHI LI; ZHI-WEN WANG. **Multi-agent based genetic algorithm for JSSP**Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826). **Anais**...IEEE, 2004Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1380676>

YANG, X. **Nature-Inspired Metaheuristic Algorithms Second Edition**. Second Edi ed.[s.l.] Luniver Press, 2010.

ZHANG, G. et al. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. **Computers & Industrial Engineering**, v. 56, n. 4, p. 1309–1318, 2009.

ZITZLER, E. Evolutionary Algorithms for Multiobjective Optimization : Methods and Applications. **English**, v. no13398, n. 30, 1999.

ZITZLER, E.; LAUMANNS, M.; BLEULER, S. A Tutorial on Evolutionary Multiobjective Optimization. In: GANDIBLEUX, P. X. et al. (Eds.). . **Metaheuristics for Multiobjective Optimisation**. Lecture Notes in Economics and Mathematical Systems. Zurich, Switzerland: Springer Berlin Heidelberg, 2004. p. 3–37.

ZUBEN, F. J. V; ATTUX, R. R. F. **Introdução à Computação Natural: apostila técnica.**Campinas, 2009.